NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A FORMAL MODEL FOR RISK ASSESSMENT IN SOFTWARE PROJECTS

by

Juan Carlos Nogueira

September 2000

Thesis Advisor: Carl R. Jones

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY	**2. REPORT DATE** September 2000	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE : A FORMAL MODEL FOR RISK ASSESSMENT IN SOFTWARE PROJECTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Nogueira, Juan C.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT

The current state of the art techniques of risk assessment rely on checklists and human expertise. This thesis introduces a formal method to assess the risk and the duration of software projects automatically. The method has been designed according the characteristics of evolutionary software processes such as productivity, requirement volatility and complexity. The formal model based on these three indicators estimates the duration and risk of evolutionary software processes. The approach introduces benefits in two fields: a) automation of risk assessment and, b) early estimation method for evolutionary software processes.

14. SUBJECT TERMS Software Engineering, Risk Assessment, Estimation Models	**15. NUMBER OF PAGES** 182
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 2-89)**
Prescribed by ANSI Std. 239-18

A FORMAL MODEL FOR RISK ASSESSMENT IN SOFTWARE PROJECTS

Juan C. Nogueira
Captain, Uruguay Navy
B.S., Universidad de la República, 1985
M.S., Universidad O.R.T., 1993

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

Author: [illegible]
Juan Carlos Nogueira

Approved by: [illegible]
Carl R. Jones, Thesis Advisor

[illegible]
Terrance Brady, Associate Advisor

[illegible]
Dan Boger, Chairman
Information Systems Academic Group

ABSTRACT

The current state of the art techniques of risk assessment rely on checklists and human expertise. This constitutes a weak approach because different people could arrive at different conclusions from the same scenario. The difficulty on estimating the duration of projects applying evolutionary software processes contributes to add intricacy to the risk assessment problem. This thesis introduces a formal method to assess the risk and the duration of software projects automatically. The method has been designed according the characteristics of evolutionary software processes such as productivity, requirement volatility and complexity. The formal model based on these three indicators estimates the duration and risk of evolutionary software processes. The approach introduces benefits in two fields: a) automation of risk assessment and, b) early estimation method for evolutionary software processes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CAPS	Computer-Aided Prototyping System
CASE	Computer-Aided Software Engineering
CMM	Capability Maturity Model
COBOL	Common Oriented Business Language
COCOMO	Constructive Cost Model
CPM	Critical Path Method
KLOC	Thousands of Lines of Code
PERT	Program Evaluation and Review Technique
PSDL	Prototype System Description Language
RAD	Rapid Application Development
SEI	Software Engineering Institute

ACKNOWLEDGMENT

First and foremost, I wish to express my sincere gratitude to the Navy of Uruguay for sending me to this exceptional School and for all the support provided to my family and myself.

Second, I wish to acknowledge my advisors. I wish to express my gratitude to Dr. Carl R. Jones for leading and orienting me through this Master in Information Technology Management, and foremost, for the confidence he showed opening the doors for a Ph.D. in Software Engineering. I also wish to thank LtCol Terrance Brady, who was my first instructor, for his advice and support.

Finally, I wish to express thanks to Dr. Lawrence Putnam. Dr. Putnam, a former NPS alumni and author of the famous model, gave me all his support when I was in the middle of confusion. I sent a mail to his company trying to clarify ideas, without too much hope of response, I must confess. Surprisingly, I did receive a response. And it was from Dr. Putnam himself! The clear insights and experience from someone who really knows about models were very helpful. And again for my surprise, a week later I received a box with his books and papers. Beau geste, Dr. Putnam. I will be always thankful.

I. INTRODUCTION

A. THE INMATURITY OF SOFTWARE ENGINEERING

"Despite 50 years of progress, the software industry remains years--perhaps decades--short of the mature engineering discipline needed to meet the demands of an information-age society." (Gibbs, 1994). Many researches have treated the problem using different approaches: formal methods, prototyping, software processes, etc. However, in the author's opinion, this assertion remains true today.

Experience suggests that building and integrating software by mechanically processable formal models leads to cheaper, faster and more reliable products (Luqi, 1997). Software development processes such the Hypergraph model for software evolution (Luqi, 1997), or the Spiral model (Boehm, 1988), have improved the state of the art. However, in the author's opinion they share a common weakness: risk assessment.

In the software evolution domain, risk assessment has not been addressed as part of the graph model. In the various enhancements and extensions, the graph model does not include risk assessment steps, hence risk management remains as a human-dependent activity that requires expertise.

On the evaluation of the spiral model, one of the difficulties mentioned by Boehm was: "Relying on risk-assessment expertise. The spiral model places a great deal of reliance on the ability of software developers to identify and manage sources of project risk." "...Another concern is that a risk-driven specification will also be people-dependent." (Boehm, 1988).

What is the reason that software engineering does not reach the maturity level of other forms of engineering? Maybe is easier to find the answer looking at the differences between software engineering and other disciplines. In the author's opinion, one difference is that software engineering is highly dependent on people. A second difference is that software engineering is younger (forty years versus centuries for civil engineering). The third difference is that the product, software, is intangible. It is difficult to estimate its real value until late in the development process. All these differences make software development projects have a great deal of uncertainty.

Many researchers (Boehm, 1898), (Charette, 1997), (Gilb, 1988), (Hall, 1997), (Jones, 1994), (Karolak, 1996), (SEI, 1996) have addressed the problem of risk assessment following the perspective of the traditional disciplines. The tools for risk assessment are guides of practices, checklists, taxonomies of risk factors and few metrics. All these methods work fine IF there is a human educated on risk assessment AND with enough experience. Such resources are very scarce. Maybe that is the reason why software engineering is still immature.

B. THE ESTIMATION PROBLEM

Since the creation of the first computers, tremendous progress has been made in terms of hardware. The general-purpose computer has been especially important because of its versatility. The stored program allows specialized applications created by software. These applications have grown in size and complexity covering all kinds of human activities. Unfortunately, the ability to build software has not followed the same rate of progress (Hall, 1997. pp xv). Gerald Weinberg said *"to call software development an infant discipline is not a moral judgement, but merely a colorful way to summarize its short history and present existence."* (Gilb, 1977. Foreword). Software engineering is the discipline that focuses on the planning, developing and maintaining software products. It seems that the creation of software imposes different challenges than the creation of hardware. In the previous section was discussed the author's hypotheses about this issue.

As the range of computer applications has grown as well as their complexity, the cost of software development has become the main cost component on a system. Literature shows that in the industry as well as in government environments, schedule and cost overruns are tragically common (Luqi, 1989). Developing software is still a high-risk activity. Despite the advances in technology and computer-aided software engineering (CASE) tools, little progress has been done in improving the management of software development projects (Hall, 1997). The acquisition and development communities, both governmental and industrial, lack a systematic way of identifying, communicating and resolving technical uncertainty (SEI, 1996). Research shows that 45 percent of all the causes for delayed software deliveries are related to organizational issues (vanGenuchten, 1991). Software is also the main cost contribution factor in computer systems (Boehm, 1981), (Karolak, 1996).

This research is focused on software project risk assessment, in other words, the prediction of success of the project. The only way to evaluate the degree of success of a project is: a) to compare the planned and actual schedules; b) to compare the planned and actual costs; and c) to compare the planned and actual product characteristics. An emergent branch of software engineering has focused on this last part: software reliability. However, the author's thinks that more emphasis must be done in the first two.

It is know that in software development, manpower and time are not interchangeable (Brooks, 1974). It is also know that productivity rates are highly variable, and that function and size are highly correlated with errors and duration of the project (Putnam, 1980). It is also learned that during the requirements phase the majority and most costly errors are introduced (Boehm 1981). It is also known that life cycle manpower patterns follow heavy tailed curves (Putnam, 1980, 1992, 1996, 1997), (Boehm, 1981). It is known some practices and heuristics that improve the development process (Humphrey, 1989). There exist CASE tools that improve the productivity. There exist macro models able to estimate with different degree of success the effort and duration of software projects (Albrecht, 1979), (Boehm, 1981, 2000), (Putnam, 1997). What it is not available is a model of the internal phenomenology of the software life cycle. Without such knowledge, risk assessment is almost impossible.

C. RESEARCH QUESTIONS

The software process is a set of activities with dependency relationships that occur over a certain period of time. From this point of view software projects do not differ from any other type project. At the beginning there exists a great deal of uncertainty that can be reduced to two types: time and money. Other intermediate metrics such as performance, rate of errors and effort can be converted to time and cost. As time goes by the level of uncertainty usually decreases as consequence of the availability of information. Unfortunately, the main resources (time and budget) also experiment the same behavior. So managers, as decision-makers, must choose between making early decisions with lots of uncertainty or postponing decisions trading time for information. This leads to the basic research question addressed in this thesis:

What are the early automatically collectable measures from the software process that describe project risk?

The concept of early measure is emphatized because recognizing the risks in the early phases increases the probability of contingency, improving consequently the competitive advantage. The research focuses on automatically collectable measures because risk identification should not impose significant extra workload and must be as objective as possible. And this leads to the second question:

How can these measures be related in order to assess project risk?

D. GENERAL APPROACH

Despite the improvements introduced in software processes and automated tools, risk assessment for software projects remains as an unstructured problem dependent on human expertise (Bohem, 1988), (Hall, 1997). It is the author's intention to find ways to transform risk assessment into a structured problem. Solving the risk assessment problem with indicators measured on the early phases would constitute a great benefit to software engineering. It is at that moment that changes can be done with less impact on the budget and schedule. The requirements phase is the crucial stage to assess risk because: a) it has a huge amount of human interaction and communication that can be misunderstood and be source of errors; b) errors introduced at this phase are very expensive to fix; c) the existence of generation tools diminishes the errors in the development process if the requirements are correct; and d) requirements evolve introducing changes and maintenance along the whole life cycle.

It is necessary to construct a model to assess risk based on measurable objective parameters that can be automatically collected and analyzed. One of the goals of this research is to integrate a risk assessment model to the previous research in Computer-Aided Prototyping System (CAPS) at the Naval Postgraduate School. This integration is required in order to capture metrics automatically and to provide project managers with a more complete tool.

Software risk management includes the identification, assessment and mitigation of risks. It requires dealing with complexity and to assign scarce resources in the most efficient way. The scope of this thesis is limited to risk identification and risk assessment. In the author's opinion, it is in these two phases where an automated method can provide major impact.

This thesis studies project risk assessment decomposing it into three classes: resource risk assessment, process risk assessment, and product risk assessment. There exists a dependency between these classes of risk. The successful use of the resources depends on their own characteristics and in the success of the product and the process. The success of the process depends on itself as well as in the success of the resources and the product. And the success of the product depends on itself and on the success of the resources applied to the process. The three classes have strong coupling and in seem to be different facets of a same entity: the project.

The measure of project risk can be viewed as the probability of developing the required product on the planned schedule and within the budget with the available software process and resources. The hypothesis is that the probability distribution is a heavy-tailed distribution probably from the Weibull family.

It is necessary to create a set of metrics customized to the characteristics of software evolution including complexity, requirements stability, personnel stability and productivity. The details of the model and the metrics are described on Chapter III. The approach has a fundamental implication: in order to assess risk it is necessary to assess duration and effort.

E. SOFTWARE EVOLUTION FOCUS

Studies have shown that early parts of the system development cycle such as requirements and design specifications are especially prone to errors (Boehm, 1981). Problems originating in the early stages often have a lasting influence on the reliability, safety and cost of the system. Evolutionary prototyping offers an iterative approach to requirements engineering to alleviate the problems of uncertainty, ambiguity and inconsistency inherent in the process. Moreover, prototyping can improve the capture of change in requirements and assumptions during the development process. This effect is particularly observed in projects involving multiple stakeholders with different points of view (Ramesh, 1995), (Conklin, 1988).

Evolutionary driven CASE tools for computer-aided prototyping provide logical assessment of the consistency and clarity of requirements and specifications. The use of prototypes facilitates the requirement phase in any type of software projects. Particularly, in real-time applications where severe time constraints impose more challenges, the use of prototypes facilitates to describe the requirements in a clear, precise, consistent and executable format. Prototypes can be applied to demonstrate system scenarios to the affected parties as a way to: a) collect criticisms and feedback that are sources for new requirements; b) early detection of deviations from users' expectations; c) trace the evolution of the requirements; and d) improve the communication and integration of the users and the development personnel.

The benefits of prototyping are unquestionable. All modern life cycle models such as Bohem's Spiral, Luqi's Graph Model, Rapid Application Development (RAD), etc. are based on prototyping. Experience suggests that building and integrating software by mechanically processable formal models leads to cheaper, faster and more reliable products (Luqi, 1997). Also, all software development processes mentioned before rely on human expertise to identify, assess, and control risk.

A second concern in the use of prototypes is that they impose a problem to project planning because the uncertain number of cycles required in constructing the product. For the most part the project management and estimation techniques are based on linear layouts of activities. Critical Path Method (CPM) and Program Evaluation Review Technique (PERT) are not well suited to deal with cycles because they are based on acyclic digraphs.

F. GENERAL RESEARCH DESIGN

The research design of this thesis is based on two approaches. First, an extended literature research will provide the basis for background and theoretical foundations covering the following topics: software engineering, software reliability, decision theory, statistics, probability, project management, and risk management.

Second, the problem of risk assessment will be analyzed using causal analysis to identify the risk factors. A model will be constructed, calibrated and validated in three ways: a) internal consistency proved by mathematics and statistics; b) black box validation by comparing its outputs in duration and effort with other available models; and c) black box validation against simulations conducted with VitéProject.

G. ORGANIZATION OF THESIS

This thesis is organized in six chapters. The introduction has been covered in the present chapter. Chapter II presents the theoretical foundation and background on software engineering, software evolution and risk management. The conceptual framework of the model is developed on Chapter III. Chapters IV and V present the detailed research design and findings respectively. Finally, in Chapter VI discusses the conclusions and future research.

II. THEORETICAL FOUNDATION

A. THEORETICAL FOUNDATION FOR SOFTWARE EVOLUTION

1. The Graph Model

The graph model is a data graph model for evolution that records dependencies and supports automatic project planning, scheduling, and configuration management. The evolution process is represented by a graph that at any given moment models the current and the past state of the software system.

Evolutionary prototyping offers an iterative approach to requirement engineering to alleviate the problems of uncertainty, ambiguity and inconsistency inherent in the process. Moreover, prototyping can improve the capture of change in requirements and assumptions during the development process. This effect is particularly notorious in projects involving multiple stakeholders with different points of view.

Computer Aided Prototyping System (CAPS) is a CASE tool that provides a collection of techniques and languages for computer-aided prototyping, including logical assessment of the consistency and clarity of requirements and specifications. CAPS methods involve the use of real-time constraints and abstract modeling to describe the requirements in a clear, precise, consistent and executable format. Prototypes can be applied to demonstrate system scenarios to the affected parties as a way to: a) collect criticisms and feedback that are sources for new requirements; b) early detection of deviations from users' expectations; c) trace the evolution of the requirements; and d) improve the communication and integration of the users and the development personnel.

Real time systems present special difficulties in terms of requirement engineering. Some requirements are difficult for the user to provide and for the analysts difficult to determine. The best way to discover these hidden requirements is via prototyping. CAPS is a tool specially suited for this task. It has a graphical, easy to understand, interface that maps to a specification language, which in turns generates Ada code. The main components of CAPS are:

(a) The prototype system description language (PSDL).

(b) User interface based on a graphic editor with a palette of objects that include operators, inputs, outputs, data flows and operator loops. A search engine helps the designer to find reusable components.

(c) The software database system provides a repository for reusable PSDL components.

(d) The execution support system consists of a translator, scheduling mechanisms, execution monitors, and a debugger.

The prototyping process consists of prototype construction and modification (evolution) based on evolving requirements and code generation. Both construction and modification are exploratory activities with a common target: to satisfy multiple users with different and often conflicting points of view. Requirement engineering is a consensus driven activity in which mechanisms for conflict resolution and traceability of requirement evolution represent critical success factors.

PSDL is based on data flow under real-time constraints and uses an enhanced data flow diagram that includes non-procedural control and timing constraints. PSDL serves as an executable prototyping language at a specification or design level. The user interface contains a graphic editor, a browser to view reusable components, and an expert system that provides the capability to generate English text descriptions of PSDL specifications.

The software database system provides the repository facilities for reusable components, as well as control of versions. The execution support system consists of a translator that generates code that binds the reusable components, scheduling mechanisms, and a debugger.

The model views a software evolution process as a partially ordered set of steps. Steps represent activities required to produce the system. A step has states that reflect the dynamic progression of the activity from the moment that it is proposed to the moment it is completed or abandoned.

The graph model has experienced its own evolution process. (Luqi, 1989) introduced the primitive version of the model. (Mostov, 1989), (Mostov, 1990) and (Luqi, 1990) refined and elaborated the model. In (Luqi, 1990), the notion of hypergraph was introduced to realize automated software evolution in multidimensional phases. Further refinements including scheduling and team coordination, were introduced by (Badr, 1993). Conflict resolution of requirements and criticisms introduced by (Ramesh, 1992) and (Ibrahim, 1996). (Luqi, 1997) extended the graph model to a hypergraph that improved the traceability of dependencies and introduced the concept of hyper-requirements. Finally, Harn extended the model to a relational hypergraph model (Harn, 1998a, Harn, 1998b, Harn, 1998c).

2. Conflict Resolution Model

Evolutionary software development requires a way to solve the conflicts that could occur between various users' points of view. System design must follow a deliberation process that involves the resolution of issues or concerns that must be addressed to satisfy user requirements. (Conklin, 1988) introduced IBIS model and (Ramesh, 1992) extended it addressing the following concepts:

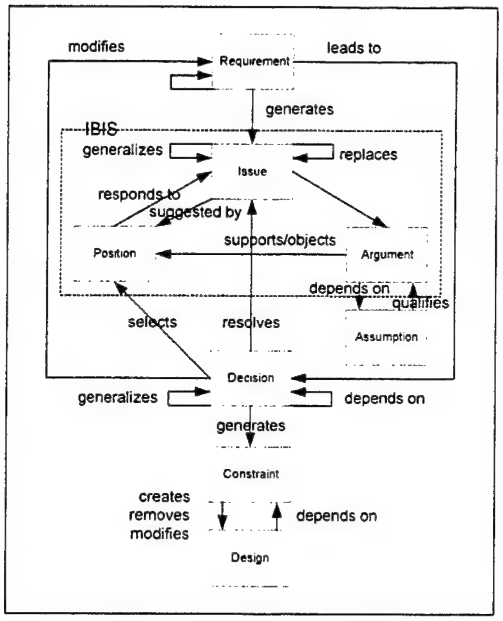


Figure 2.1: Ramesh's model

(1) Requirements represent the goals to be satisfied by the design process.

(2) Issues are questions or concerns that different stakeholders introduce.

(3) Positions are alternatives that address an issue.

(4) Arguments either support or object a position.

(5) Decisions represent the resolution of issues and lead to constraints.

3. Relational Hypergraph Model

The relational hypergraph model was introduced in (Harn, 1999e) is a formal model for software evolution that incorporates the features of the previous graph models. The hypergraph model (Luqi, 1997) represents the evolution history, as well as the plan for the future, in a hypergraph. A hypergraph is a directed graph with hyperedges, which may have multiple input and output nodes. The formal definition of the relational hypergraph model is presented on Appendix A.

4. Conclusions about the Relational Hypergraph Model

The precedent definitions constitute the formal specification of the relational hypergraph model. It constitutes a framework to support software evolution processes. However, risk assessment has been omitted as part of the specification. This issue creates a human dependency in risk assessment. Despite this limitation, the model can be extended to support automated risk assessment.

B. THEORETICAL FOUNDATION FOR RISK MANAGEMENT

1. Risk and Uncertainty

Developing software is still a high-risk activity. Despite the advances in technology and CASE tools, little progress has been done in improving the management of software development projects. The acquisition and development communities, both governmental and industrial, lack of systematic way of identifying, communicating and resolving technical uncertainty (SEI, 1996). Research shows that 45 percent of all the causes for delayed software deliveries are related to organization issues (vanGenuchten, 1991). Software is also the main cost contribution factor in computer systems (Boehm, 1981), (Karolak, 1996). Besides the improvements in tools and methodologies, there is not evidence of success in moving from the idea to the product. A study published by the Stadish Group reveals that the number of software projects that fail has dropped from 40% in 1997 to 26%. However, the percentage of projects with costs and schedule overruns grow up from 33% in 1997 to 46% (Reel, 1999).

Part of the problem is the misinterpretation the importance of risk management. It is usually viewed as an extra activity layered on the assigned work, or worst, as an outside activity that is not part of the software process (Hall, 1997), (Karolak, 1996).

A second source of the problem is the lack of tools needed to perform risk management (Karolak, 1996). The main reason for this lack of tools is that risk assessment is apparently an unstructured problem. To define unstructured problems it is necessary to explain previously structured processes. Structured processes involve routine and repetitive problems for which a least one solution exists. Unstructured processes require decision-making based on a three-phase method (intelligence, design, choice) (Turban & Aronson, 1998). An unstructured problem is one in which none of the three phases is structured. Risk management is highly biased by manager's perceptions and characteristics that are difficult to represent in an algorithm. Depending on the decision-maker's risk behavior, he/she can opt to choose early with lack of information, or to postpone the decision gaining time to invest in obtaining information, but loosing opportunity control.

The third source of the problem is the confusion created by the informal use of terms. Often, the software engineering community (and most part of the project management community (Wideman, 1992)), use the term "risk" in a very lax sense. Generally, software risk is viewed as a measure of the likelihood of a loss or an unsatisfactory outcome affecting the software from different points of view: project, process and product (Hall, 1997), (SEI, 1996). This definition of risk is misleading because confounds the concepts of risk and uncertainty. In general, most part of the decision making during the software process is under uncertainty rather than under risk. Let discuss briefly the decision-making environments in order to clarify these concepts.

There are three possible situations in any decision context: certainty, risk and uncertainty. Decisions under certainty occur when the decision-maker knows exactly the consequence of each alternative or decision choice. In this case the decision process is very simple: the alternative with the best outcome is chosen by examination. However, this is a rare situation.

Usually the decision-maker does not have a complete picture of the future, but knows the probability of occurrence of the various states of nature. In this case the decision-making is under risk, and many techniques can be addressed to support the decision: expected monetary value, expected value of perfect information, opportunity loss, sensitivity analysis among others (Render, 1997). All these methods rely on the huge hypothesis of knowing the exact probability for each scenario.

A completely different situation is when the decision-maker does not have the precise information about the probabilities of occurrence of the different states of nature, or the list of the states. In this case it is impossible to assess the outcome, hence a completely different set of techniques must be applied to support the decision-making process: maximin, minimax, Laplace, Hurwicz, or minimax regret (Render, 1997).

The distinction between these two concepts is important for decision making because it leads to drastically different approaches to risk assessment:

(a) Assessing software risk using a probabilistic approach usually measuring reliability. In this case the decision-making is under risk. However, even using probabilistic models there exist a component of uncertainty created by uncertainties in parameter values, uncertainties in modeling, and ambiguities in the degree of completeness (Baybutt, 1989).

- Ambiguities in parameter values are consequence of the need to estimate parameter values from data. The ambiguities arise because the available data is usually incomplete and because the analyst makes inferences from a state of incomplete knowledge.
- Deficiencies of model in representing the reality.
- Completeness ambiguities are introduced by the inability of the analyst in evaluating exhaustively all contributions to risk.

The treatment of uncertainties in risk analysis involves the evaluation of uncertainties in the input, the propagation through each part of the risk analysis, the combination of the uncertainties in the output, the display and interpretation of risk estimates, and the treatment of uncertainties in decision-making.

(b) Assessing software risk using a framework of practices and guidelines (SEI, 1996). In this case there is not a probabilistic model to rely on, hence the decision-making is under uncertainty.

It follows, as it was previously stated, that the largest part of the decisions made by software-managers are under uncertainty. Two categories of research attacked the issue from different angles. First, probabilistic approaches have been made with success to assess the reliability of the product (Lyu, 1995), (Schneidewind, 1975), (Musa, 1998).

However, these approaches assess software reliability when it is too late for software engineering purposes, because the product is complete or almost complete.

A second category of research has addressed the problem from a different perspective, trying to assess the risk in parallel with the development process. However, in this case the approach is less rigorous and unstructured, basically the proposals are lists of practices and checklists (SEI, 1996), (Hall, 1997) or scoring techniques (Karolak, 1996). Paradoxically, SEI defines software technical risk as a measure of the probability and severity of adverse effects in the development of software that does not meet its intended functions and performance requirements (SEI, 1996). However, the term "probability" in this case is misleading, because the probability is unknown. There is a third category of research focused mainly on estimation of effort and time that has characteristics of both previous groups. This approach tangentially related to risk and will be discussed in Section E.

The fourth source of confusion is introduced when the term "risk" is used to describe different things. It is not only erroneously used as a synonym of uncertainty as stated before, but it is also used as a synonym of "threat" (SEI, 1996), (Hall, 1997), (Karolak, 1996). In this research the term risk is reserved to indicate the probabilistic outcome of a succession of states of nature, and the term "threat" is used to identify the dangers that can occur.

2. Decision under Uncertainty

Very frequently decision-makers make decisions using incomplete information. Particularly, the problem of decision-making under uncertainty involves choosing among a set of alternatives under the following conditions:

- The outcome of each course of action depends on several possible states of nature.
- The outcome for each alternative under each state of nature is known.

- The probability of occurrence of each state of nature is unknown.

When the probability of occurrence of each state of nature is unknown or cannot be assessed, then the following five techniques can be applied:

- Maximax criterion. This criterion implies an optimistic vision of the future. The method consists in choosing the alternative that maximizes the maximum outcome for every alternative.
- Maximin criterion. This method finds the alternative that maximizes the minimum outcome. It is a pessimistic approach.
- Laplace criterion. This method uses equal probabilities for each state of nature and then computes the outcomes for each alternative, choosing the higher outcome.
- Criterion of realism. This method is also known as Hurwicz criterion. It is a compromise between an optimistic and a pessimistic decision. The decision-maker must choose a coefficient of realism α between 0 and 1. This coefficient is applied to the favorable state of nature outcome, and $(1 - \alpha)$ is applied to the outcome of the unfavorable state of nature. The alternative with the higher weighted sum is chosen.
- Minimax criterion. This method is based on opportunity loss. It finds the alternative that minimizes the maximum opportunity loss within the alternatives.

3. Subjective Probabilities and Utility Theory

Another way to deal with uncertainty situations is to use a subjective estimation of the probabilities of occurrence of the different states of nature. This approach is easy to implement but requires a great deal of experience to judge the success probability of each alternative. Group consensus techniques, like Delphi method (Dalkey & Helmer, 1963), are usually very helpful in such situations (Marshall, 1995).

Decisions trees are based on the expected monetary value (EMV) could lead to bad decisions in many cases. There are many situations in which a linear payoff function is unable to represent the behavior of people (Marshall, 1995). These are the two reasons to study utility theory. In practice, historical data can be analyzed to obtain an objective estimate of the outcomes. But in situations, especially those that incorporate management decisions, historical data could be not relevant. The judgments and beliefs of the decision-makers may be more important that estimating relevant probabilities (Marshall, 1995). Before describing utility theory in detail, two definitions are required:

> "The indifference probability for a decision problem between a risky venture and a riskless alternative with given known results is that probability of success in the risky venture for which the decision-maker is indifferent to the two alternatives." (Marshall, 1995).
>
> "The certainty equivalent to a risky venture is the least amount the decision-maker would have to obtain for certain by choosing the riskless alternative." (Marshall, 1995).

In many situations the indifference probability and the certainty equivalent would have different values for different people. The differences reflect various behaviors toward risk. Utility assessment assigns the worst outcome a utility of 0 and the best outcome a utility of 1. All other outcomes have a utility value between 0 and 1. When two or more alternatives are equally attractive (or unattractive), that is the decision-maker is indifferent, then their utility value should be the same. The problem is to find the probability that makes the decision-maker indifferent.

Until now, it was considered decision-making with only one attribute. A more general scenario would have many attributes for measuring the decision. Often, these attributes conflict with each other, hence optimizing one results in suboptimizing others. Thus, it is necessary to use trade-offs to resolve such conflicts. A common approach to solving multiattribute problems is to combine the different measures into a single numeric measure. The problem can then be treated as single attribute problem (Marshall, 1995). In many decision problems it is very difficult to establish measurement criteria.

Particularly, when the decisions are not at the operational level. At the operational level, decisions can be measured in terms of lines of code or function points. However, at the project management level, the effectiveness of a decision could be measured in terms of quality, stability, marketing impact, etc. In such cases, multiattribute utility theory should be applied (Fig. 2.2).

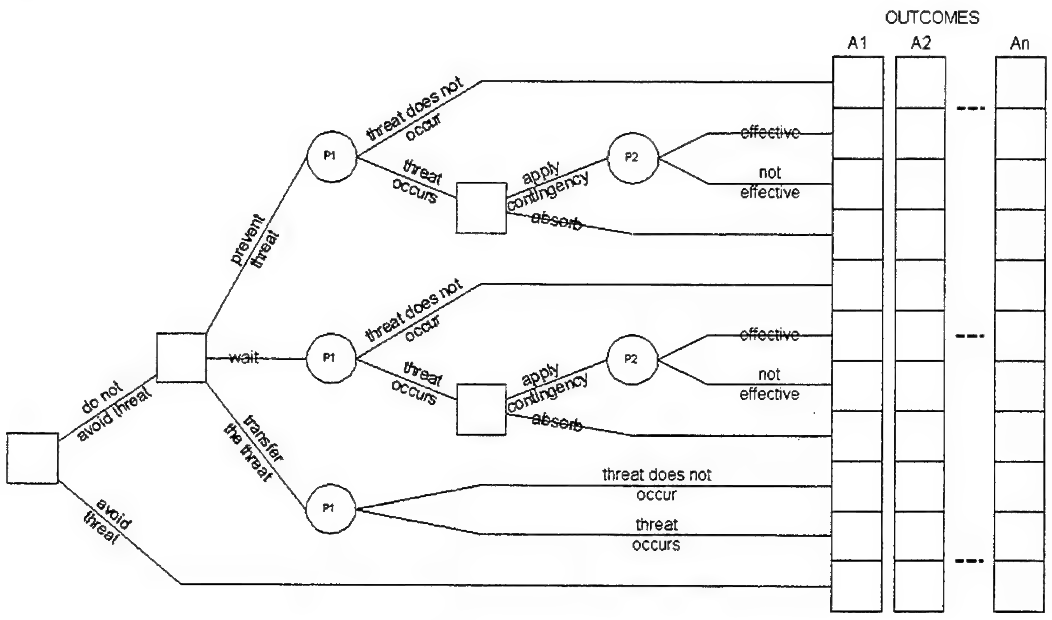


Figure 2.2: Multiattribute Decision Tree

The decision-maker must provide his estimation of return for each attribute related to the decision, as a vector R = (R1, R2, ..., Rn). The decision-maker must introduce also his preferences as a weight vector W = (W1, W2, ..., Wn). The outcomes of each attribute are given by Ai, such:

$$Ai = Wi * Ri$$

$$\text{where } \sum_{i=0}^{n} Wi = 1$$

The outcome for each alternative is then calculated as a function of the sum of the attributes (A1, A2, ..., An) converted to a value between 0 and 1, where 1 is given to the best outcome and 0 to the worst.

C. SOFTWARE ENGINEERING FOUNDATIONS

The literature and research about risk and risk management is very wide. This research focuses on a partition that comprehends operational research, project management, software engineering and software reliability. Operational research provides the theoretical foundation to describe and analyze risk. Project management, software engineering and software reliability apply the theory. This research narrows the problem to software, specifically to the software engineering domain.

Taxonomies are very useful. They facilitate the understanding of complexity by partitioning the problem in disjoint pieces that are simpler. The review of the literature shows two different schools of thinking:

(1) The group that studies the problem of software risk from the point of view of the development process. This group follows a forward approach managing the risk in parallel with the development process. The caveat of many of these approaches is that they are not formal and their success mainly depends on human expertise.

(2) The group that studies the problem of software risk from the point of view of reliability. This group follows post mortem approach, studying the product created and inferring its future behavior. This category is strongly supported by statistics. However, from the point of view of software engineering, it has less impact because the findings arrive too late to make changes in the product without incurring in huge costs.

1. Software Engineering Institute (SEI)

The Software Engineering Institute (SEI), at Carnegie Mellon, relies on improving the process as a way to improve the products and diminish risks. This philosophy is particularly clear in a guideline created as a request of the USAF by SEI and Mitre Corporation (Humphrey, 1987). The document describes a method to assess the software engineering capabilities of contractors. The guideline stated that the quality of the product depends on the quality of the process, which depends on the technology used to support it, which depends on the maturity level of the organization. Hence, by transitivity, the quality of the product depends on the maturity level of the organization. Consequently, assessing the maturity of the organization it is possible to estimate the attributes of the product

The SEI proposes a three dimensional vision of risk management process composed by (SEI, 1996):

(a) Temporal dimension that includes the micro perspective, that is from the point of view of the project, and the macro perspective that covers the complete life cycle.

(b) Methodological dimension that includes practices (software risk evaluation (SRE), continuous risk management (CRM) and team risk management (TRM)), and basic constructs including the SEI's risk taxonomy.

(c) Human dimension that consider the perspectives of the individual, the team, the management and the stakeholder.

The SEI approach to risk assessment uses a risk taxonomy questionnaire to ensure that all risk areas are systematically addressed. The complete taxonomy can be reached on (SEI96). Table 2.1 presents a brief summary to show the characteristics analyzed.

Table 2.1: SEI's taxonomy of risks (SEI, 1996)

1. ***Product engineering***
   - 1.1. *Requirements (stability, completeness, clarity, validity, feasibility, precedent, and scale).*
   - 1.2. *Design (functionality, interfaces, performance, testability, hardware constraints, and non-developmental software).*
   - 1.3. *Code and unit test (feasibility, testing, coding/implementation).*
   - 1.4. *Integration and test (environment, product, system).*
   - 1.5. *Engineering specialties (maintainability, reliability, safety, security, human factors, and specifications).*
2. ***Development environment***
   - 2.1. *Development process (formality, suitability, process control, familiarity, and product control).*
   - 2.2. *Development system (capacity, suitability, usability, familiarity, reliability, system support, and deliverability).*
   - 2.3. *Management process (planning, project organization, management experience, program interfaces).*
   - 2.4. *Management methods (monitoring, personnel management, quality assurance, and configuration management).*
   - 2.5. *Work environment (quality attitude, cooperation, communication, and morale).*
3. ***Program constraints***
   - 3.1. *Resources (schedule, staff, budget, and facilities).*
   - 3.2. *Contract (type of contract, restrictions, and dependencies).*
   - 3.3. *Program interfaces (customer, associate contractors, subcontractors, prime contractor, corporate management, vendors, and politics).*

The SEI approach presents some problems:

- Many of the items covered by this taxonomy are highly subjective and difficult to express in terms of equations. How to measure politics? How to measure with confidence the morale? The only way is to use qualitative measures that have inherent subjectivity.
- Many of the items are covered more than once. As instance human factors, work environment and budget seem to be highly related.
- The guidelines are sets of heuristics and good practices which impact, on the success of the project, depends on human experience.

Consequently, this approach relies on the ability of the human using the checklist. It is required an expert to assess the risk.

2. Hall

Elaine Hall's method for managing risk (Hall, 1997) is derived from the SEI model. In her view four major critical success factors are responsible for risk management: People, Process, Infrastructure, and Implementation (P^2I^2).

- People participate in risk management by implementing the processes according to the plans, by detecting problems, communicating issues and introducing uncertainties in their work. People at all levels need to be educated, involved, and motivated in risk management.
- Process must transform uncertainties into risks. The transformation is based on identifying the sources of risk, analyzing the risk based on some established criteria, planning alternative strategies for risk resolution, tracking the risk metrics, and resolving the risk triggering action plans. Unfortunately, how to do the transformation (that is the key problem), is not addressed in (Hall, 1997) nor in (SEI, 1996).
- Infrastructure establishes the culture that supports risk management.
- Implementation is the execution of the plans, assigning responsibilities, authorities, tools and methods.

On Hall's method, checklists based on SEI taxonomy, work breakdown decomposition, meetings, reviews, and surveys are the tools for risk identification. All these tools are human dependant and highly unstructured. Hence, the method is very difficult to automate. However, Hall emphasizes the use of metrics to identify occurrence of risks such as progress in milestones, size (LOC), change (requirements added, changed, deleted), quality (number of defects), staff (turnover) and risk exposure. Risk analysis, risk planning, risk tracking and risk resolution are based on planning, and a set of resolution techniques and tools inherited from SEI's model. Hall's approach has the same problems of SEI's model.

3. Charette

(Charette, 1997) introduced the concept of risk management in maintenance. The author states that during maintenance, risk management is more difficult than during development. First, maintenance projects provide more opportunities for risk and less freedom to mitigate it as a consequence of the previous version of the system. Second, it involves more attention to customer related issues. The approach is based on uses SEI's taxonomy as the tool to identify treats and SEI's software risk evaluation process to assess the risk. Charette's approach has the same problems that previously addressed about SEI's model. The method relies on human experience.

4. Jones

During the 60's and the 70's, IBM have focused significantly on software processes. Many technologies were invented in IBM's laboratories: HIPO diagrams, joint application design, formal inspections, structured walkthroughs, integrated cost and estimation tools, and formal specifications. It is significant also that CMM has characteristics that can be traced back to IBM when Humphrey was at IBM. Neither SEI's CMM or Software Productivity Research (SPR) (Jones, 1994) addresses how to solve the problems of estimation. SPR is a software process introduced by Capers Jones that has some very similar characteristics with CMM. Jones and Humphrey were working at IBM during the seventies, so it is not surprising that both models have common characteristics. As an example the five-level scale of CMM correspond to the five-scale of SPR. (Jones, 1994) observed those significant risks are not the same across all software domains. He introduced six categories of software projects with different kinds of risks. Table 2.2 shows the percentage of projects at risk for each category. Note that the table is ordered showing on the top the risk factors more common for all the projects categories.

Table 2.2: Jone's top risk factors (Jones, 1994)

Risk factor	MIS	Embedded	COTS	Military	Outsource	End-user
Low quality	60%	50%	55%	45%		65%
Schedule	65%	70%	50%	75%		
Creeping user requirements	80%			70%	45%	
Excessive paperwork		60%		90%		
Cost estimates	55%	65%				
Low productivity				85%		
Non-transferable applications						80%
Inadequate documentation			70%			
High maintenance costs					60%	
Inadequate configuration control	50%					
Friction between personnel					50%	
Acceptance criteria					30%	20%
Maintenance problems						50%
Redundant applications						50%
Competitors			45%			
Cancellation		35%				
Litigation expense			30%			
Legal ownership of deliverables					20%	

Jones stated that the ten most serious risk factors observed in the SPR assessments are:

(1) Inaccurate metrics. The generalized use of LOC as a productivity metric introduces errors because the differences in the languages and programming styles. Counting LOC does not address the complexity involved in recursion nor object-oriented paradigm. LOC is very difficult to estimate during the requirements. Albrecht addressed this problem with the introduction of function points. However, recently Kitchenham,

Kemerer and others have introduced some criticisms to this metric. This issue will be discussed on Chapter III.

(2) Inadequate measurement. Data collection is not always correctly done, even in the case of cost collection. One major leak in terms of cost is the work of end users.

(3) Time pressure introduced by irrational schedules or by continuously changing requirements. This second factor is more intense as the complexity of the systems grows. Projects with more than 1000 function points are most likely to experience this problem.

(4) Management weaknesses due to lack of education in estimation, planning, measuring and assessment.

(5) Inaccuracies in cost estimation. Despite the numerous commercial software tools available, the use of estimation tools is not generalized.

(6) Naive belief that moving to a new technology will create improvements in productivity or quality.

(7) Late requirements. Even with the availability methodologies like prototyping, JAD or QFD, and metrics like function points or feature points, which permit to understand the impact of changes, late requirements continue to be a major threat.

(8) Low quality. The current average of defects per function point in U.S. is 5 defects per function point.

(9) Low productivity. The current U.S. average for military projects is about 3 function points per man-month. For MIS the productivity is about 8 function points per man-month.

(10) Cancellation of projects is directly proportional to their size. This particularly critical above 10,000 function points or 1 million LOC.

The contribution of Jones reveals some common threats characteristics of different types of software projects. It is particularly significant the impact of paperwork and low productivity in DoD projects. The caveat of this work is that it does not provide a method to manage risk relying on the experience of the project manager to make the right decisions.

5. Karolak

(Karolak, 1996) introduced a classification scheme that divides the risk in three software-risk elements: Technical, Cost and Schedule. This model uses subjective Bayesian probability approach to assess software risks. Each of the three software risk elements are influenced by ten risk factors according with Table 2.3:

Table 2.3: Karolak's scheme (Karolak, 1996)

	Software Risk Element		
Software Risk Factor	Technical	Cost	Schedule
Organization	LOW	HIGH	HIGH
Estimation	LOW	HIGH	HIGH
Monitoring	MEDIUM	HIGH	HIGH
Methodology	MEDIUM	HIGH	HIGH
Tools	MEDIUM	MEDIUM	MEDIUM
Risk culture	HIGH	MEDIUM	MEDIUM
Usability	HIGH	LOW	LOW
Correctness	HIGH	LOW	LOW
Reliability	HIGH	LOW	LOW
Personnel	HIGH	HIGH	HIGH

(a) "Organization" addresses risks associated with the maturity of the organization structure, functions, management and communications.
(b) "Estimation" addresses the risks associated with inaccuracies in estimating resources, schedules and costs.
(c) "Monitoring" refers to risks associated with identifying problems.
(d) "Methodology" addresses the risks associated with the lack of formal methodology and standards.
(e) "Tools" refers to the risks associated with the development tools.
(f) "Risk culture" addresses the characteristics of the management decision-making style.
(g) "Usability" refers to risks associated to the software product after it is delivered.
(h) "Correctness" addresses to the risks associated with compliance with requirements after the delivery.

(i) "Reliability" refers to the risks of failures after the delivery.
(j) "Personnel" includes the risks associated with the knowledge and skills of the development team.

The key element to identify and measure risks on Karolak's approach is a questionnaire used to evaluate the risk factors (81 questions: organization 8, estimation 7, monitoring 7, methodology 7, tools 9, risk culture 11, usability 6, correctness 9, reliability 12, and personnel 12). The answer for each question in a number between 0 and 1, where 0 represents none and 1 represents all. The main contribution of this model is that it can be automated; indeed Karolak developed a tool called SERIM (Software Engineering Risk Model). However, the problem with this approach is that even though the tool provides support, human experience is still required as the key factor to identify risks.

6. Project Management Institute (PMI)

The Project Management institute (PMI) introduced a methodology for risk management (Wideman, 1992) generalized for any kind of projects. The method is based on four phases: risk identification, risk assessment, risk response, and documentation. Risk identification follows an informal approach based on taxonomies, expert's opinions and workgroup techniques. The assessment phase may range from subjective evaluation to the use of metrics. This phase includes also the analysis of impact. On this model there are two planning activities: response planning, and contingency planning; and three typical risk response strategies: avoidance, deflection, and absorption. PMI uses the term risk to denote two different concepts: the probability of occurrence of a threat and the threat itself. Another terminology issue in this approach is the use of the term risk in scenarios which decisions are made under uncertainty rather than risk. The approach is too general to be useful in software engineering.

7. Mitre Corporation

Mitre Corporation developed a Web application (RAMP) to capture risk management experience and retrieve experiences from other projects and advice. The user introduces the characteristics of his project in a static HTML form. A query is launched over the RAMP databases creating a dynamic HTML form with a set of projects with similar characteristics. The user can select one or more of these projects and a second script retrieves risks from the database. The result of this second query is a report containing links to the applicable documents (Garvey, 1997). This approach helps the decision-maker providing him of related documents about similar projects, but it did not release the need of human experience to manage risk.

8. Rockwell

At Rockwell, an improvement on communicating risks more effectively resulted the following benefits: predictable program performance, better reviews, improved process, and improvements in management practices. Three key elements are the cause of successful risk management at Rockwell: repeatable process, widespread access to adequate knowledge and functional behavior (defined as human factors).

Functional behavior implies human interactions, motivations and incentives, perceptions and perspectives, communication and consensus, and decision making and risk tolerance. (Gemmer, 1997) identified the following functional behaviors: a) manage risk as an asset, b) treat decision making as a skill, c) active seek for risk information, d) seek diversity in perspectives and information sources, e) minimize uncertainty on time, control and information, f) recognize and minimize bias in perceiving risk, g) plan for multiple futures, h) be proactive, i) improve the decision-making skills, and j) reward who identify and manage risks early.

Gemmer identified the following causes for risks: a) uncertainty in time, b) uncertainty in control, and c) uncertainty in information. Risk management is usually an uncertainty scenario characterized by: a) uncertainty in the impact or consequence, b) there exists a time frame to prevent or mitigate, c) there exists a coupling or domino effect, d) there exists uncertainty about the probability distribution function (Gemmer, 1997).

9. Boehm

Boehm has been studying the problem of risk management for more than a decade. His contributions to the area are notable. He introduced the importance of verification and validation of software requirements and design specifications during early phases of the project as a way to mitigate risk (Boehm, 1984). Such activities include: a) completeness, b) consistency, c) feasibility, and d) testability of the specifications. Completeness implies that all the documents and references exist and that there are no missing items, functions or products. Consistency is both internal and external, and implies traceability. Feasibility requires validate that the project can be achieved with the actual resources, that it will satisfy the users' needs, that it will be maintainable, and estimate the risk. Testability requires unambiguous and quantitative specifications.

Boehm introduced the Spiral model (Boehm, 1988) as a substitute to the Royce's Waterfall model. The Spiral model was the first software process in which risk assessment was a driving factor. The author recognized however that there exist difficulties in applying his model: a) matching the evolving process with contracts; b) relying on risk-assessment expertise, the model is people dependent in terms of identification, management and risk-driven specification; c) the need of further elaboration in the spiral steps (Boehm, 1988); d) ambiguities about how to initiate, terminate and iterate within the spiral; e) complexities in handling incremental development such as refinements from previous versions; f) difficulties in formalize processes; and h) some steps result more complex than were envisioned (Boehm, 1988a).

In (Boehm, 1989) and (Boehm, 1991) he introduced a method for risk management (Table 2.4). Risk management is divided in two families of activities: risk assessment and risk control.

Table 2.4: Boehm's classification (Boehm, 1991)

Risk assessment is decomposed into:

(1) Risk identification by use of checklists, decision driver analysis, assumption analysis, and decomposition.

   a. Checklist (top 10 risks)
      - Personnel shortfalls.
      - Unrealistic schedules.
      - Requirement risks.
      - Developing the wrong functionality.
      - Developing the wrong user interface.
      - Developing extra functionality not essential or with marginal usefulness.
      - Continuous stream of requirement changes.
      - Problems in external components.
      - Problems in external tasks.
      - Performance shortfalls. Straining computer science capabilities (trying to do more than the possibilities of the state of the art technology): distributed processing, AI, human-machine interface, algorithm speed and accuracy, computer security, reliability and fault tolerance.

   b. Decision driver analysis:
      - Politically driven decisions.
      - Marketing driven decisions.
      - Applying the wrong solution to the problem because there exist compromises or preferences. (Story of the guy that was looking for his keys in the night. He was looking in a different spot were he presumably lost the keys, but this spot was under a light).
      - Short-term versus long-term decisions.

c. Assumption analysis.
   - Comparison with previous experience.
   - Pessimistic approach (Murphy's Law).

d. Decomposition
- Pareto 80-20 phenomena.
- Task dependencies (high fan-in implies risk: if anything slips the project aborts. High fan-out also implies risk: if the precondition slips then the effect is in many parts of the project).
- Uncertainty areas in the plan.

(2) Risk Analysis:
a. Decision trees.
b. Network analysis using PERT and probabilistic network analysis.
c. Cost risk analysis using COCOMO, Putnam or other estimation tool for effort and duration.
d. Automated analysis tools (PROMAP, PROSIM, RISNET, SLAM, Opera/Open Plan, PRISM, REP).

(3) Risk Prioritization:
a. Assess the risk probabilities from historical data, Delphi or other group technique.
b. Deal with compound risks.
c. Deal with triggered risks (dominoes effect).

<u>Risk control is decomposed into</u>:
(1) Planning.
(2) Resolution.
(3) Monitoring (milestone tracking and top-10 risk tracking).

Boehm alerted that current approaches to the software process make have tendency to make high-risk commitments. "The waterfall model tempts to over promise software capabilities in contractually binding requirements specifications before analyzing the implications. The evolutionary development makes too easy to introduce new ideas and requirements that can lead to a disaster." (Boehm, 1991). Recently in an article coauthored with De Marco they showed a pessimistic and pragmatic stating "doing software risk management makes good sense, but talking about it can expose you to legal liabilities. If a software product fails, the existence of a formal risk plan that acknowledges the possibility of such a failure could complicate and even compromise the producer's legal position." (Boehm, 1997).

Boehm's contributions to risk management are multiple. This research picked the most important ones such as the Spiral model, his analysis of the activities required for risk management, and his risk management method. Due to its relevance, a separate section includes the discussion about the Constructive Const Model (COCOMO). Despite his contributions, Boehm recognizes that the issue of relying on humans to assess risk remains unsolved. The use of checklists, decision driver analysis, assumption analysis, and decomposition is not enough to automate risk identification and assessment.

10. McFarlan

McFarlan introduced a model to assess risk on information system projects based on a three-dimensional checklist covering the three major dimensions influence the risk inherent in a project: a) project size in terms of budget, staffing levels, elapsed time and number of departments affected; b) experience with the technology; c) project structure in terms of definition of the tasks and deliverables (McFarlan, 1974). The importance of his contribution resides in the identification of different facets on software projects. This model relies on checklists and in the experience of the decision-maker to evaluate risk.

11. Gilb

In his classical text on Software Engineering Management (Gilb, 1988) presented a set of principles or rules of engagement with risk. The approach is informal. Gilb's principles are heuristics that were the state of the art at that time. His work was included because he was a pioneer in recognizing the problem and the need of being proactive.

12. USAF

(USAF, 1988) defines risk as the probability at a given point in a system's life cycle that predicted goals couldn't be achieved with the available resources. Due to the high degree of uncertainty high precision is not useful during the early phases. As the system progresses the uncertainty is transformed into risk, therefore higher precision is required. The USAF introduced a method to abate risk based on checklists and estimations of probability of occurrence and effects. They decompose the software risk in four dimensions: performance, support or maintainability, cost, and schedule. The effects on the project are categorized into catastrophic, critical, marginal and negligible. The four risk dimensions are measured in terms of their probability of occurrence and their effect according to Table 2.5.

Table 2.5: USAF scheme for risk

Prob.	1.0 - 0.7	0.7 - 0.4	0.4 - 0.0	0.0
Impact	Frequent	Probable	Improbable	Impossible
Catastrophic	HIGH			
Critical		MODERATE		NONE
Marginal				
Negligible			LOW	

The USAF method is very simple and robust. However, it is informal, relying on checklists and experience of the evaluator.

D. ESTIMATION MODELS

In this section presents three models to estimate effort and duration so software projects: COCOMO, Putnam and function points. The importance of these estimation models resides in that constitute a preliminary approach to assess risk.

1. The COCOMO Family

COCOMO (for Constructive Cost Model) was introduced by (Boehm, 1981) is a family of models constituted by Basic, Intermediate, and Detailed COCOMO. Basic COCOMO is an easy to calculate model applicable to small to medium software projects. Intermediate COCOMO is based on the Basic model and includes effort adjustment factors. The detailed COCOMO accounts the influence of additional factors on individual project phases. These earlier models are known as COCOMO 81.

Projects are classified into three categories: a) organic which are characterized by small size, small teams and low environmental noise; b) embedded characterized by strong complex coupling with hardware or other kind of tight constraints like real time systems; and c) semidetached which are intermediate between the previous two categories. The details of the model can be found in (Boehm, 1981), but it is important to highlight the following assumptions that show the optimistic bias of the model.

- The development period considered by COCOMO 81 starts at the beginning of the design phase. The requirements phase is not covered.
- The estimation covers only the direct-charged labor. Costs related to computer center operators, secretaries, higher management, and support are excluded.
- The model assumes that a man-month is 152 hours of working time.
- The model assumes that the project will enjoy of good management.
- Finally, the model assumes that the requirements will remain unchanged.

The input parameter for COCOMO 81 is the size estimation in thousands of lines of code (KLOC), which constitutes a drawback because of the difficulty of predicting the size during early stages. COCOMO II addresses the problem of size estimation introducing a more abstract indicator of size called object points (a variation of function points). This model is being calibrated.

2. Putnam

In the 50's, Peter Norden from IBM developed a manpower model. He used the following curve of the Weibull distribution family, named after the 19th century physicist Lord Rayleigh:

y	$= K (1 - \exp(-at^2))$, and its first derivative
y'	$= 2 K a t \exp(-at^2)$, where
y	= cumulative percentage of total effort
y'	= manpower rate in terms of people per unit of time
K	= effort in men-unit of time
t	= development time
a	= a constant governing the time to manpower peek.

During the 70's, Putnam, an alumni of the NPS, introduced a model applying the concepts developed before by Norden at the IBM development laboratory of Poughkeepsie. This model is supported by a commercial tool named SLIM (Software Life Cycle Management). The use of the Rayleigh curve as a reasonably good fit for the manpower distribution has been proved by Norden, (Putnam, 1980) and (Boehm, 1981). Putnam observed that there exist a strong correlation between lines of code and schedule, manpower and defects. He recognized differences in terms of development difficulties between real time systems and normal information systems (Putnam, 1980 and 1996). Putnam's model is based on the following assumptions (Londeix, 1987):

- A development project is a finite sequence of purposeful, temporally ordered activities, operating on an homogeneous set of problem elements, to meet a specified set of objectives.
- The number of problem elements is unknown but finite.
- Problems are detected, recognized and solved applying effort.
- The occurrence of problem solving follows a Poisson process.
- The number of people working in the project is proportional to the number of problems ready to solve at that time.

The main equation of this model relates the size of the project in lines of code to the effort and the schedule:

S	$= C_k K^{1/3} t_d^{4/3}$, where
S	= number of delivered source instructions
K	= life-cycle effort in man-years
t_d	= development time in years
C_k	= a "technology constant"

The required development effort (DE) is estimated as 40% of the life-cycle effort. That is:

$$DE = 0.4\,K = 0.4\,(S/C_k)^3\,(1/t_d^4).$$

One difficulty of the approach, as with COCOMO, is the requirement of knowing the number of lines of code at the beginning of the project. Putnam suggests the use of the Delphi method to estimate S.

Let a = minimum size estimation,
b = most likely size,
c = maximum size estimation.
The estimator of the expected size, $E(S) = (a + 4b + c) / 6$.
And the estimator of the standard deviation is $s = (c - a) / 6$.

Another difficulty is to estimate the technology constant C_k. Putnam suggests deriving it from previous projects. That is, analyzing post-mortem projects with known S, K and t_d it is possible to derive the value of C_k. This approach introduce two constraints:

- To apply the model it is required to have available historic data.
- The development process must be repeatable, that is at least CMM level 2.

(Boehm, 1981) states that this method is not good for projects employing incremental development, but this comment could be a little biased. Nevertheless, changes in requirements lead to a new estimation. According to its author, the method is not precise for small projects with development time of two years or less. This seems to be caused to a more rectangular manpower pattern observed in small projects. The method has been verified with more than 4000 projects. (Conte, 1986) observed also that the model works "reasonably well" on very large systems but overestimates the effort on medium and small ones. Other criticisms of the same authors are exaggeration of the effect of time compression on the development effort, excessive weight on the size, and excessive sensibility to changes of the technological constant.

During this research an experiment was conducted to compare Putnam's model with COCOMO 81. The experience consisted in comparing the estimates of 100 projects with sizes from 10KLOC to 1MLOC using Basic COCOMO for organic, semidetached and embedded systems with Putnam estimation. To avoid problems of tuning, the effort in Putnam was based using the average of the development times of COCOMO. Similarly, the time in Putnam was calculated using the average to COCOMO efforts. Both cases used a constant of technology = 10100 as suggested in (Boehm, 1981). The following graphs show the findings:

- In terms of effort Putnam's model is almost the average of embedded and semidetached COCOMO (Fig. 2.3).
- In terms of development time, the models are quite similar, being Putnam's estimation more optimistic (Fig. 2.4).

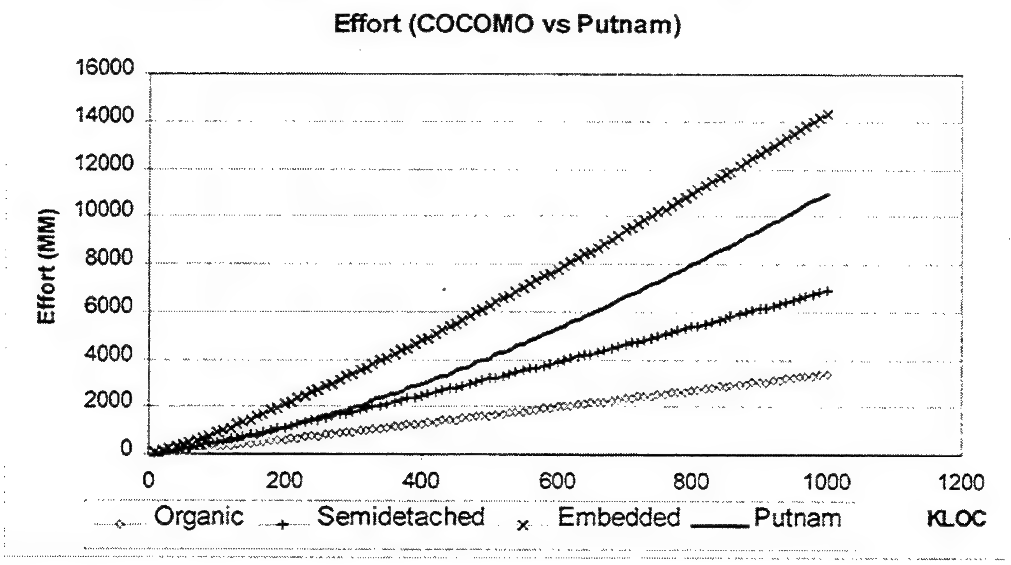


Figure 2.3: Effort Estimated Using COCOMO and Putnam Models

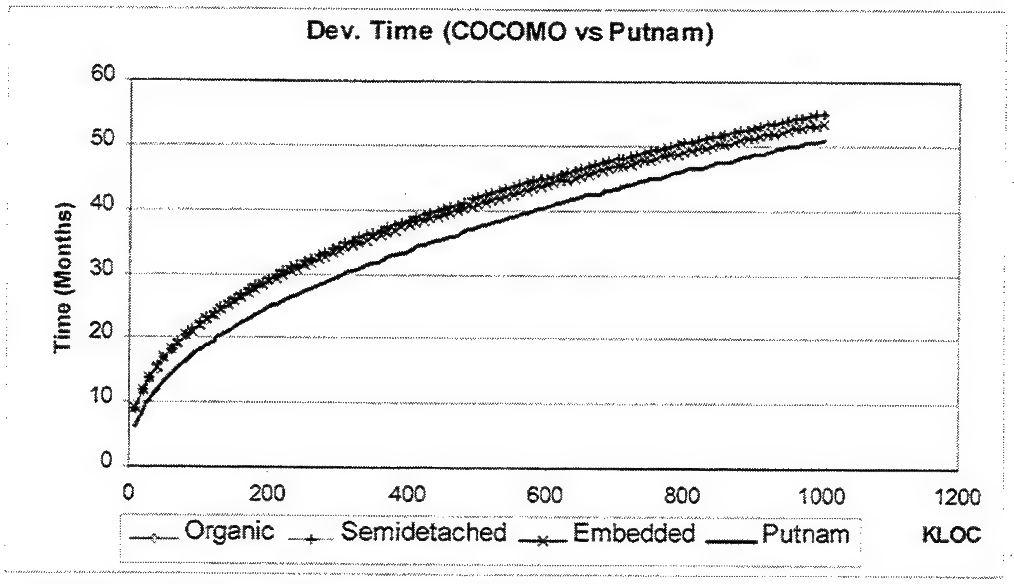


Figure 2.4: Development Time Estimated Using COCOMO and Putnam Models

3. Function Points

Functional complexity has been studied for years because it is highly correlated with effort and risk. The traditional functional complexity metric has been introduced by (Albrecht, 1979 and 1983). Function Points had an enormous success because:

- It is an early metric. It can be calculated after the preliminary analysis of the system.
- It is easy to calculate. There are only five input parameters to compute and fourteen fine-tuning adjustments, and the whole process can be done manually.
- It was the first metric that related complexity to number of lines of code.

The procedure for calculating Function Points is quite simple. It is required to count the number of inputs, outputs, queries, files, and system's interfaces. Each of the five parameters is classified into simple, medium or complex. Depending on the parameter and its complexity the count is multiplied by a weight factor. Table 2.6 presents the template for the calculation.

Table 2.6: Function Points Calculation (Albrecht, 1983)

	Simple	*Weight*	*Medium*	*Weight*	*Complex*	*Weight*	*Total*
Inputs	(	* 3) +	(	* 4) +	(	* 6) =	
Outputs	(	* 4) +	(	* 5) +	(	* 7) =	
Queries	(	* 3) +	(	* 4) +	(	* 6) =	
Files	(	* 7) +	(	* 10) +	(	* 15) =	
Interfaces	(	* 5) +	(	* 7) +	(	* 10) =	
					NAFP	=	Σ

The result of the total is called Not Adjusted Function Points (NAFP). Fourteen adjustment factors, whose values are in the range of zero to five, describing the environment are added. Finally the Function Points are calculated by the formula:

$$FP = NAFP * (0.65 + 0.01 * \Sigma F_i)$$

where NAFP is the non adjusted Function points

F_i is each of the fourteen adjustment factors

Despite its attractive approach, Function Points has many weaknesses. First of all, the metric was derived from a study of MIS projects in the seventies. Today, there are many issues that are not considered by the metric and that are contributors to complexity. For instance, recursive functions, reuse, inheritance, communication by messages, and polymorphism are not covered by the metric. The languages have evolved also, and differ a lot from the COBOL of the seventies. Programming styles experienced a dramatic change that does not appear in the metric.

(Kemerer, 1993) reported some weaknesses of the metric. Similar results have been reported by (Kitchenham, 1993) and (Kitchenham, 1997). The main issue is that function points is a not well-formed metric because there is a correlation between their constituent elements. In her conclusions she stated that:

- The individual function point elements were not independent.

- Not all the function point elements are related to effort.
- An effort prediction metric based on inputs and outputs is just as good predictor as function points.
- An effort prediction metric based on the number of files and the number of outputs was only slightly worse that Function Points.
- To get good estimates it is necessary estimation methods and models based on the organization's performance, working practices, and software experience.
- Uncertainty and risk cannot be managed effectively at the individual project level, but in the organization context. If a single project had to ensure against all possible risks and uncertainty, its cost would be prohibitive. The sources for estimate uncertainty are: a) measurement error caused because some of the input variables in a model have inherent accuracy limitations; b) model error caused because no estimation model can include all the factors involved; c) assumption error caused because some of the hypotheses about input parameters are incorrect; and d) scope error caused because the project under study is outside the model's domain.

Even if there was evidence of defects in the metric, nobody introduced a better alternative. So, function points remained for many years as the most common prediction metric. More recently, some extensions to function points have been introduced such as "feature points" and "Boeing's 3-F function points" addressing the effort estimation for embedded systems.

4. Conclusions about COCOMO, Putnam and Function Points

All these methodologies have some weaknesses with respect to software evolution. First, the need of a size estimate as an input parameter limited the applicability of COCOMO and Putnam methods. Second, the characteristics counted on function points are quite different than the specification attributes. Third, the criticisms introduced by (Kemerer, 1993; Kitchenham, 1993 and 1997) suggested that despite the correlation observed between complexity and size, other metrics could be more accurate on this metric.

E. MODERN PROJECT MANAGEMENT TECHNQUES: VitéProject

VitéProject is a modeling and simulation tool that integrates the organizational work of projects explicating the interdependencies between tasks and roles not only from the point of view of producer-consumer such as in CPM or Pert, but also communication and rework dependencies. VitéProject is the commercial version of VDT (Virtual Design Tool), a research based on contingency theory directed by Dr. Raymond Levitt at Stanford (Jin, 1996).

CPM models are sequential interdependencies through explicit representation of precedence relationships between activities. This simplified vision of the project cannot address the dynamics created by reciprocal requirements of information in concurrent activities, exceptions management, and the impacts of actor interactions.

The original model of VDT was based on the following observations about collaborative, multidisciplinary work in large complex projects:

- Organizational tasks in the project can be divided into two categories: production work that directly adds value to the product, and coordination work that facilitates the previous one.
- Contingency theory provides qualitative insights about the extent of coordination work, but did not provide information about how to address the bottleneck problems created by coordination.

The model integrates the micro level description of the entities that perform work and process information called "actors". Actors can be individuals or small teams acting as a unique and cohesive unit where individuals are not differentiated. Actors have two basic behaviors: *attention allocation* and *information processing*. As a consequence of such behaviors, actors perform *production* and *coordination*. The model is based on the following assumptions:

- Actor allocation assumption: Each actor has one input buffer where all the incoming information and requests for production or coordination work arrive. The input buffer is a queue that supports different policies: priorities, FIFO, and random. Each actor has also an output buffer to place its accomplished work.
- Actor capacity allocation assumption: An actor has certain information-processing capacity determined by its skill type, skill level, and allocable time. An information processing work can be processed and completed if the actor allocates sufficient capacity to the job. This assumption implies: a) information processing requires not only attention but also takes time; b) the information content of a work is related to the skills; c) the volume of a work is related to the time; d) actors have limited capacity to allocate.
- Actors cannot allocate 100% of its capacity to work because they are interrupted by: a) information requests from other actors; b) decision-making to solve exceptions produced by subordinate actors; c) meetings; and d) processing noise, that is all other interruption created outside the project that have impact on the actor.

The organization structure is modeled through simulation. The organization variables such as control structure, communication structure, formalization and matrix strength, influence actor's micro level actions, and consequently an organization's emergent performance appears. The use of complex adaptive systems to model organizational behavior has been discussed also by (Brown, 1998).

F. ORGANIZATIONAL THEORY

This section introduces some foundations of organizational theory that support the research. Why to review the organizational foundation if the research is about software engineering? First, software development requires teamwork, more specifically organized work. So it is required to understand the dynamics of organizations as artificial social entities that exist to achieve a specific purpose, in this case to develop software. Second, organizations are made up of individuals who accomplish diverse desegregate activities that require coordination and consequently information exchange. These two activities, despite their impact, have not been covered by the research in estimation models. Third, VitéProject is customized for general projects. In order to obtain a rigorous simulation, it is required to customize the tool according to the characteristics of software engineering.

1. Introduction

As software systems increased in complexity, software development evolved form a primitive art into software engineering. Methodologies and software tools were developed to help development processes. Most of the present tendencies (DOD-STD-2167A, ISO-9001, SEI/CMM) try to standardize processes, emphasizing planning and structure (Humphrey, 1990). Some authors criticize those approaches stating that they underestimate the dynamics of the software development (Bach, 1994), (Abdel-Hamid, 1997). Others question that activities such as research and development are not addressed by TQM principles (Dooley et al., 1994). In the author's opinion, many of the problems on current software projects have organizational roots. This view is also supported by (van Genutchen, 1991)[1] and (Capers Jones, 1994)[2]. The typical software engineering process is a succession of decision problems trying to transform a set of fuzzy expectations into requirements, specifications, designs and finally code and documentation. The traditional waterfall software process failed to accomplish their purpose because it applied a method valid for well-defined and quasi-static scenarios.

[1] Van Genuchten found that 45% of all the causes for delayed software are related to organizational issues.

[2] Capers Jones found that on military software developments the two more common threats are excessive paperwork (90% of the time) and low productivity (85% of the time).

This hypothesis is far from the reality. Today, modern software processes (Boehm, 1988)(Luqi, 1989) are based on evolution and prototyping. These approaches recognize the fact that software development presents an ill-defined decision problem and they fail in assessing automatically the risk. In the author's view, software development projects present special characteristics that require to be solved in order to achieve an improvement in the state of the art. These particularities affect the **strategic planning**, the **organizational structure**, and the **engineering applied to software**. In these three areas chaos theory can provide clues for possible solutions.

(Woodward, 1965) has studied the relationship between technological complexity and structure, classifying the technology into three types: a) unit (custom made and non-routine jobs), b) mass (large batch or mass production in assembly lines), and c) process (highly controlled, standardized and continuous processing such as refineries). This scheme was created for the manufacturing industry and it is not very suitable for software engineering. However, it has some characteristics of unit and process technologies: high proportion of skilled workers, low formalization and low centralization.

Perrow (Burton, 1998) introduced a two-dimensional classification of the technology (Fig. 2.5). The first dimension is the analyzability of the problem varying from well-defined to ill-defined. The second dimension is the task variability, which means the number of expected exceptions in the tasks. The scheme lacks of a third dimension representing time. Hence, in this projection, software engineering occupies part of the non-routine and part of the engineering regions. During the earlier phases of the development usually the problem is ill-defined. That is why the requirements phase is so prone to errors. After several

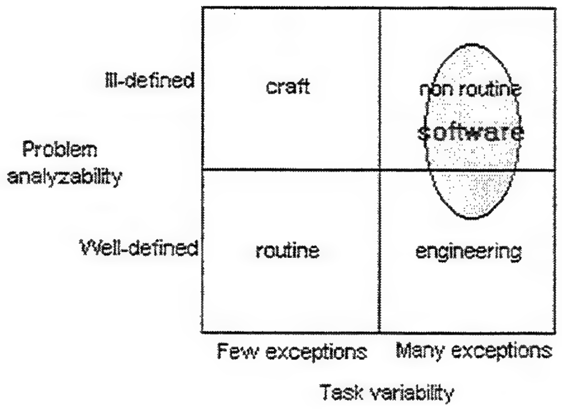


Figure 2.5: Perrow's classification

prototypes and evolution cycles the problem is transformed into well-defined and the system can be specified. This is a key difference with other forms of engineering already discussed in the first chapter. Highly skilled personnel, low formalization and centralization, high information processing demand, and coordination obtained through meetings characterize the organizations in this region.

A second line of research (Burton and Obel, 1998 pp. 174-180), introduced a classification based on four-variable model: equivocality, environmental complexity, uncertainty and hostility. Equivocality is "the existence of multiple and conflicting interpretations", it is a measure of the lack of knowledge or the level of ignorance whether a variable exists in the space. Uncertainty is the lack of knowledge about the likelihood of values for the known variables. Environmental complexity is the number of factors in the environment affecting the organization and their interdependency. Finally, hostility is "the level of competition and how malevolent the environment is." In Table 2.7, the fourth variable, hostility, was disregarded because when hostility grows over a certain threshold, it overrules other factors (Burton & Obel, 1998 pp. 177). In highly hostility scenarios only a highly centralized organization ("regular army"), or a low-formal-low-complex organization ("guerilla") are the possible alternatives.

Table 2.7: Burton & Obel's scheme (Adapted from Burton & Obel, 1998 pp 181-182)

Equivocality	Enviromental Complexity	Uncertainty	Formalization	Organizational Complexity	Centralization
Low	Low	Low	High	Medium	High
Low	Low	High	Medium	High	Medium
Low	High	Low	High	Medium	Medium
Low	High	High	Medium	High	Low
High	Low	Low	Medium	Medium	High
High	Low	High	Low	Low	High
High	High	Low	Medium	Medium	Low
High	High	High	Low	Low	Low

Software development scenarios usually correspond to high equivocality that decreases over time, high environmental complexity and high uncertainty scenarios (dark gray in Table 2.7), which corresponds to low formalization and low organizational complexity, with centralization inverse to the environmental complexity. The recommended organization could be ad hoc or matrix with coordination by integrator or group meeting. The information exchange is rich and abundant. The incentive policy should be based on results. These parameters constitute the key points to customize the behavior matrix of VitéProject to software developments.

2. The Edge of Chaos

Chaos theory describes a specific range of irregular behaviors in systems that move or change (James, 1996). Chaotic does not mean random. The primary feature distinguishing chaotic from random behavior is the existence of one ore more attractors. Without the existence of such attractors the quasi-chaotic scenarios could not be repeatable. It is important to realize that a chaotic system must be bounded, nonlinear, non-periodic and sensitive to small disturbances and mixing. A system that has all these properties can be driven into chaos. The edge of chaos is defined as "a natural state between order and chaos, a grand compromise between structure and surprise" (James, 1996). The edge of chaos can be visualized as an unstable partially structured state of the universe. It is unstable because it is constantly attracted to the chaos or to the absolute order. Usually people have the tendency to think that the order is the ideal state of nature. This could be a big mistake. Research on organizational theory (Stacey, Nonaka, Zimmerman); Management (Stacey, Levy); and economics (Arthur) support the theory that operation away from equilibrium generates creativity, self-organization processes and increasing returns (Roos, 1996).

Change occurs when there is some structure so that the change can be organized, but not so rigid that it cannot occur. Too much chaos, on the other hand, can make impossible coordination and coherence. Lack of structure does not always mean disorder. Let illustrate this idea with an example. A flock of migratory ducks in a lake has little structure. However, a few minutes after they start flying some order appear and the flock creates a V-shape formation. This self-organized behavior occurs because a loose form of structure exists. Experiments with intelligent agents governed by three rules (a) try to maintain a minimum distance from the other objects in the environment, including other agents; b) try to match the speed of other agents in the vicinity; and c) try to move toward the perceived center of mass of the agents in the vicinity), show the same macro behavior. Independently of the starting position of the agents, they always end up in a flock. Even if an obstacle disturbs the formation, the pseudo-order is recovered some time later. This self-organized behavior emerges despite the absence of leadership and without an explicit order to form a flock.

A more interesting example is the behavior of software development teams. A recent article (Cusumano, 1997), describes the strategies of Microsoft to manage large teams as small teams. Dr. Cusumano says "What Microsoft tries to do is allow many small teams and individuals enough freedom to work in parallel yet still function as one large team, so they can build large-scale products relatively quickly and cheaply. The teams adhere to a few rigid rules that enforce a high degree of coordination and communication." This seems to be a description of the emerging behavior in a complex adaptive system. It is self-adaptive because the agents realize the adjustment to the environment, and it is emergent because it arises from the system and can only be partly predicted. As in the example of the ducks, a few rules of interaction between the agents (in this case people) generate a efficient behavior. The three rigid rules at Microsoft are: a) developers integrate their work daily forcing the synchronization and testing of the work; b) developers responsible for bugs must fix them immediately, and are responsible for the next day integration; and c) milestone stabilization is sacred. Another possible explanation of Cusumano's observations could be the presence of an underlying structure that propitiates the creativity and productivity.

Complex adaptive systems, as the one just described, are made up with multiple interacting agents. The emergence of the complex behavior requires some conditions. The first condition is the existence of more than one agent. A second condition is that agents must be sufficiently different to each other that their behavior is not exactly the same in all cases. When agents behave exactly the same way exhibit predictable, not complex, behavior. Finally, a third condition is required. Complex adaptive behavior only occurs in the edge of chaos.

3. Some of the Risks of Being in the Edge of Chaos

Limiting the structure in organizations can be useful in situations when innovation is critical or when is required to revitalize bureaucracies. However, if the structure is debilitated beyond a certain minimum, it leads to an undesired state. Some traits can alert the eminence of such anarchic situation known as the "chaos trap" (Brown & Eisenhardt[3], 1999): a) emerging of a rule-breaking culture, b) missing deadlines and unclear responsibilities and goals, and c) random communication flows.

On the other hand, focusing in hierarchy and disciplined processes, emphasis on schedules, planning and job descriptions may lead to a steady inert bureaucracy. Organizations in such a state react too late failing to capture shifting strategic opportunities. This is the case of a "bureaucratic trap", where there are also some observable warning traits: a) rule-following culture, b) rigid structure, tight processes and job definitions, and c) formal communication as the only channel.

The alternative is "surfing" the edge of chaos avoiding both attractors. That requires limited structure combined with intense interaction between the agents, giving enough flexibility to develop surprising and adaptive behavior. Organizations in this state are characterized by having an adaptive culture. People expect and anticipate changes. A second characteristic is that the few key existing structures are never violated. Finally, real time communication is required throughout the entire organization.

[3] Kathleen Eisenhardt is a NPS alumni.

Being in the edge of the chaos implies an unstable position. Some perturbations can cause the rupture of this delicate equilibrium and the fall into one of the two steady states. A potential perturbation factor is the organizational collaboration style. Too much collaboration can disturb the performance of each agent and consequently, the whole system is affected. On the other hand, too little collaboration destroys the advantage of acting organized and leads to paralysis. Other sources of perturbation are the tendency to be tight to the past and cultural idiosyncrasy, or by contrary, to loose the link with the past. In one case, the change becomes impossible. In the other case, the assets from previous experiences are not capitalized. The equilibrium point is called regeneration. In such unstable state, mutation can occur. Therefore the inherited characteristics that give competitive advantage in a certain scenario can be perpetuated, and new variations are introduced. If too little variation exists, natural selection fails. This process permits that complex adaptive systems change over the time following a Darwinian pattern.

(Kauffman, 1995) introduced the concept of fitness landscape. This concept can be understood by observing the behavior of species. In the competition for survival, species attempt to alter their genetic make-up by taking adaptation trying to move to higher "fitness points" where their viability will be enhanced. Species that are not able to reach higher points on their landscapes may be outpaced by competitors who are more successful in doing so. If that occurs the risk of extinction increases. The same principle applies between predator and prey. Each development in the abilities of one species generates an improvement on the abilities of the other. This concept is called co-evolution. Certain higher fitness points have more value to some species than to others. The contribution a new gene can make to a species' fitness depends on genes the species already has. As more complicated is the genetic pattern (more evolved), the probability of conflict of a new adaptation increases slowing down the speed of variations.

Natural selection is an effective, but not generally efficient way to evolve (Brown & Eisenhardt, 1998). The process requires some amount of mutation to avoid the sudden convergence on suboptimal characteristics. Some of the characteristics lost in the past can be reintroduced being useful in the new scenario. Many errors are committed during this blind process. A more efficient way to evolve is by recombination of the pool of genes using genetic algorithms. This technique has been applied to improve the performance of robots, however the idea can be used to improve the competencies of organizations. If too much or too less variation occurs the result always conduct to the failure of the system.

4. The Strategic Planning Issue

Traditional approaches to strategic planning emphasize picking a unique strategy according to the competitive advantages of each organization. Porter's five-force approach (Porter, 1980), assumes that there exists some degree of accuracy in the prediction of which industries and which strategic positions are viable and for how long. In a high-velocity scenario the assumption of a stable environment is too restrictive. Customers, providers, competitors, and potential competitors, as well as substitute products are evolving faster than expected. The introduction of new information technology tools, the Internet and the globalization of the markets are contributing to this phenomenon, and nothing seems to reverse the process. The failure of long-term strategic planning is not a failure of management, it is the normal outcome in a complex and unpredictable environment. A growing number of consultants and academics (Santosus, 1998)(Brown and Eisenhardt, 1999) are looking at complexity theory, to help decision-makers improve the way they lead organizations.

How useful could a map of a territory that is constantly changing its topography be? In fast changing environments, survival requires a refined ability to sense the external variables. Traditional approaches rely on strategic planning and vision. However, in unstable environments planning would not be effective because it is impossible to predict the scenario's evolution in terms of markets, technologies, customer's needs, etc. Organizations relying only on one vision supported by a tight planning, risk paying little attention to the future. Consequently, their sensing organs are blind to foresight of the future. A certain amount of inertia and commitment to the plans is required to prevent erratic changes caused by reaction diverse variables.

If the time window of the opportunities is shrinking, a different form of thinking is required. The present technological situation can be described as a fast succession of short-term niches. The ability to change is the key of success for surviving in such a variable environment. In a systemic approach, the General Systems Theory establishes that organizations are systems whose viability depends on some basic behaviors (von Bertalanfy, 1976):

- Ability to sense changes in the environment. This is the most primitive form of intelligence, if it is not present the probabilities of survive are minimum.
- Ability to adapt to a new environment modifying the internal structure and behavior. The system tries to auto-regulate to survive the crisis in hostile scenarios, or take advantage of the opportunities in favorable ones.
- Ability to learn from the past, anticipating the auto-regulation behaviors and structure before the environment change. This ability requires intelligence able to infer conclusions from the past according to the context of the variables sensed on the present.
- Ability to introduce changes in the environment, making it more favorable to the system's needs. In this case, the system has developed the technology (know how and tools) to exert power over the environment.

Any mechanical or computing system has some or all of these abilities. These same abilities could be found in any form of life. The more developed the system is, the more of the above characteristics has. Darwin's Evolution Theory validates this line of reasoning. Natural selection, acting on inherited genetic variation through successive generations over the time is the form of evolution. Variation is the way used by biological systems to probe the environment presenting many alternatives, some of them ending on failure but a few very successful. This process is an inefficient but very effective way of improvement.

Experiments can provide a certain amount of knowledge about the future. In some sense, probes are mutations in small scale that can cause only small losses. The results give insights to discover new options to compete in the future and stimulate creative thinking. The research investment pays dividends when a new way of competition is discovered altering the status quo's rules. When the changes in the environment occur too fast, sensing the variables becomes more difficult. It is possible that a specialized organ was not able to react on time to record the metric and transmit the alert. In this case, the system starts to lose information threatening its own viability. When the changes in the environment are too drastic, even if the sensor organs detect the change, the inference organs may not be able to determine an effective course of action because they do not have a previous experience, or because the decision-making process requires more time. This situation also threats the viability of the system in the long run. The effects of drastic variations and high rate of change over systems can be visualized with simple experiments: a) increasing the speed of transmission in a communication channel beyond some limit will provoke the lost of part or the entire message, b) modifying the pH in the soil beyond a certain limit can cause the death of a plant. The same syndrome can be recognized in any type of organization. It is possible to employ a new strategy. "Competing on the Edge" is a new theory defines strategy as the creation of a relentless flow of competitive advantages that, taken together, form a semi-coherent strategic direction (Brown & Eisenhardt, 1998). The key driver for superior performance is the ability to change, reinventing the organization constantly over the time. This factor of success can be applied to software engineering as well as to other decision problems with similar characteristics.

If the environment is moving, like in surfing, the best way to remain in equilibrium is by being in the rhythm. Successful corporations such as Intel or Microsoft are in perpetual movement, launching new products with certain rhythm. Intel is faithful to its founder's (Moore) law: the power of the microprocessors double every eighteen months. Microsoft has a proportional pace on the software sector. The challenges imposed by hyper competition create similar characteristics than in software engineering developments. So, the rules of engagement proved effective for one discipline could result useful in the other.

5. Application in Software Engineering

Chaos in software development comes from various sources: a) the intrinsic variable nature of requirements, b) the changes introduced by new technologies, c) the dynamics of the software process, and d) the complex nature of human interaction. These conditions are sufficient for the development of complex adaptive systems where the agents are software developers or parallel collaborative projects. Software development scenarios usually have high equivocality, high environmental complexity and high uncertainty. The suggested organizational structure to deal with such scenarios (Burton and Obel, 1998) should have low formalization and organizational complexity, centralization inverse to the environmental complexity, and rich and abundant information exchange. The recommended organization should be ad hoc or matrix, with coordination by integrator or group meeting. This organizational style is difficult to achieve when the organizations are large.

A simple solution can be recognized at Microsoft (Cusumano, 1997): a) parallel developments by small teams with continuous synchronization and periodically stabilization, b) software evolution processes where the product acquires new features in increments as the project proceeds rather than at the end of a project, c) testing conducted in parallel as part of the evolution process, and d) focus creativity by evolving features and "fixing" resources. Cusumano observed that small development teams were more productive because: a) fewer people on a team have better communication and consistency of ideas than large teams, and b) in research, engineering and intellectual work individual productivity has big variance. Software development requires teamwork, more specifically organized work. So it is necessary to understand the dynamics of organizations as artificial social entities that exist to achieve a specific purpose, in this case to develop software. Such organizations are made up of individuals who accomplish diverse desegregate activities that require coordination and consequently information exchange.

In order to apply this approach three factors should be resolved. First, automated risk assessment is required (the topic of this research). Second, evolutionary software processes should establish the maximum speed of the evolution. If the evolutions occur too fast, without a period of relaxation, it is certain that the process will fall into chaos. On the other hand if the speed is too slow then the productivity could result affected. The correct rhythm for software processes has not been researched and remains on the hands of the project manager. Third, software processes should be focused on *flexibility* and *extensibility* rather than in *high quality*. This assertion sounds scary. However, it is necessary to prioritize the speed of the development over zero defects. Extending the development in order to reach high quality could result in a late delivery of the product, when the opportunity niche has disappeared. This paradigm shift is imposed by the competition on the edge of chaos.

A shift from the traditional long-term development organizations is required. Virtual teams created as temporary dynamic project-oriented structures, with a composition of skills matching exactly the objectives could improve the current performances. Such virtual organizations are not exposed to bureaucratic loads and do not require to absorb the cost of permanent staff (Senegupta and Jones, 1999). Larger developments could be achieved by parallel projects loosely coupled sharing a common architecture such CORBA or DCOM. This paradigm enables the possibility of managing large developing organizations as if they were small. In such scenarios, the benefits of complex adaptive systems will occur at two levels. At the micro level, inside each small project, the agents are individuals. Second, at the macro level where the agents are the small projects.

6. Conclusion

Complex adaptive systems appear as the most attractive way to deal with changing environments. Besides some indicators introduced by (Brown and Eisenhardt, 1999), the academic research is not mature enough to assert a methodology for competition on the edge. Some enterprises like Microsoft and Intel seem to have discovered and applied this form of strategy since many years ago, but little information has permeated. The drastic change proposed in the software processes aims to use the benefits of programming in the small to programming in the large. The quality-driven paradigm should be revised, and that the objective should be shorter delivery times, flexibility, and scalability.

III. CONCEPTUAL FRAMEWORK

This chapter contains the framework for risk identification and risk assessment. Causal analysis was used to find the primitive origins of threats in a project, trying to find a way to identify and assess risk automatically. From the point of view of software engineering, it is necessary to create the methodology to frame the decision-making process during the early stages of the life cycle, when changes can be done with less impact on the budget and schedule. According to (Field, 1997), the most significant causes of IS project failures are: lack of understanding of user's needs, ill defined scopes, poor management of project changes, changes in the chosen technology, changes in the business needs, unrealistic deadlines, user's resistance, loss of sponsorship, lack of personnel skills, and poor management.

Risk management can be divided in three activities: risk identification, risk assessment and risk resolution. Risk identification is the set of techniques designed to alert and identity possible threats. Risk assessment is the quantitative analysis of the probabilities and impacts of the identified threats. Risk resolution is the application of resources and effort to avoid, transfer, prevent, mitigate or assume the risks. This third activity is beyond the scope of this research.

In order to achieve risk management, an organization requires a minimum level of maturity that can be associated to Capability Maturity Model (CMM) level 2. SEI followers said that "many organizations are unable to manage risks effectively for any of the three following reasons: a risk-averse culture; an inadequate management infrastructure to support effective risk management; or the lack of a systematic and repeatable method to identify, analyze, and plan risk mitigation" (Carr, 1997). If an organization is not able to collect metrics, any attempt to formally identify and assess risks is impossible. Project managers require critical information to make timely and prudent decisions. It is not surprising that increased complexity can decrease a project manager's ability to identify and manage risk.

In this research vision, software risks could be controlled if the problems of how to administer uncertainty, complexity and resources are solved. Transforming the unstructured problem of risk assessment leads to a formal method able to be translated into an algorithm. In order to structure the problem, project risk was analyzed and decomposed into simpler parts. Using causal analysis three major risk factors were identified: process risk, resource risk and product risk. Each of these factors introduces risks by themselves but mainly due to the interaction between them.

Resource risk is affected by organizational, operational, managerial and contractual parameters such as resources, outsourcing, personnel, time and budget among others. The literature is abundant in this area (Hall, 1997), (Karolak, 1996), (Grey, 1995). Various approaches use subjective techniques such as guidelines and checklists (SEI, 1996), (Hall, 1997), (Karolak, 1995), which even when could be supported by metrics, require expert's opinion.

Engineering development work procedures such as software development, planning, quality assurance, and configuration management cause **process risk**. The more complex a process is, the more difficult it is to manage, and the more education, training, standards, reviews, and communication are required. Consequently, complexity grows. The software process complexity has been partially covered by research in terms of subjective assessments about maturity level and expertise (SEI, 1996), (Hall, 1998), (Humphrey, 1989). However, a more precise and objective method is required. Several approaches to study process complexity have been introduced in the field of management. Particularly, (Nissen, 1998) introduced an objective methodology that can be used to measure the complexity of processes that can be applied to software development.

Cyclomatic complexity of the process graph is another candidate metric. These two approaches measure complexity in a static way. Simulation can be used to measure the complexity of the dynamics of the processes (Abdel-Hamid, 1989 and 1991).

Finally, **product risk** is related to the final characteristics of the product, its complexity, its conformance with specifications and requirements, its reliability, and customer satisfaction. The product introduces its own risk terms of quantitative and qualitative attributes. Two basic product-risk factors, requirement stability and requirement complexity, were identified. Requirement stability is measurable using the set of metrics previously described. Due to the inherent lack of structure of requirements, it is necessary to transform them into specifications in order to compute complexity. Other product characteristics such as reliability and maintainability are not of interest to identify and assess risk on early stages. Reliability can be measured only after completion or almost completion. Maintainability can be measured only after the design stated. Both measures are useful to control the project in future phases. For instance, applying Schneidewind's model it is possible to monitor the occurrence of software errors as a predictor for future cumulative detected and corrected errors. These estimations are useful in order to: (1) identify the trade-off function between error reduction and cost of error reduction, (2) provide quantitative basis for accepting or rejecting software during functional testing, and (3) provide quantitative basis for deciding whether additional testing is warranted based on the cost of error removal. Maintainability can be measured using metrics such as introduced in (Lorenz, 1995).

The analysis showed a dependency between these classes of risk. The success of the project depends on its own characteristics and in the success of the product and the process. The success of the process depends on itself as well as the successful use of resources in the project, and the success of the product. And the success of the product depends on itself, but depends on the success of the resources and the process. The three areas constitute an equivalence relation (Fig. 3.1) because the symmetric, transitive and reflexive properties apply. Moreover, the three classes are one equivalence class in the relation. The strong dependency between the three concepts reflects the fact that resources, process and product are different facets of a same entity: the project.

The process provides the description of its environment and the theoretical requirements to execute it. Consequently, the process introduces threats due to its requirements and characteristics: complexity, technology required, budget required, schedule required, and personnel skills required. The resources represent the actual allowances in personnel, tools, budget and schedule. They impose constraints that could not match the process requirements. The productivity is consequence of the matching of these two facets of the project.

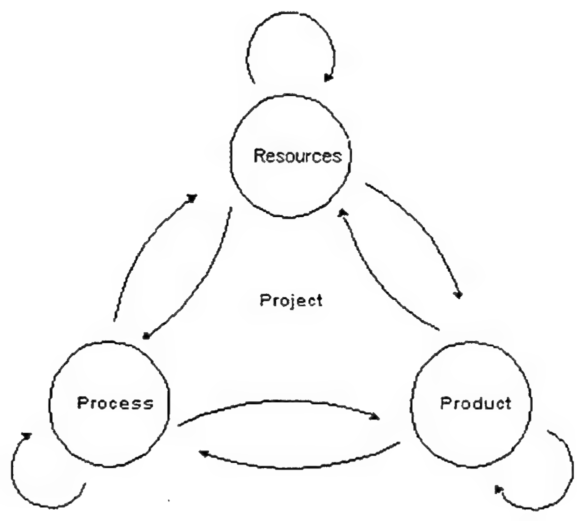


Figure 3.1: The Equivalence Relation

The decomposition created by causal analysis (Fig. 3.2) reveals:

- A method to identify risks by comparing the degree of mismatching between the product and process characteristics, against the resource constraints.
- Candidate indicators to be used in the estimation model. In Chapter V, three groups of metrics will be introduced: a) for requirements, b) for personnel—the key resource—, and c) for complexity. These three groups of metrics correspond to the three risk factors identified by causal analysis.

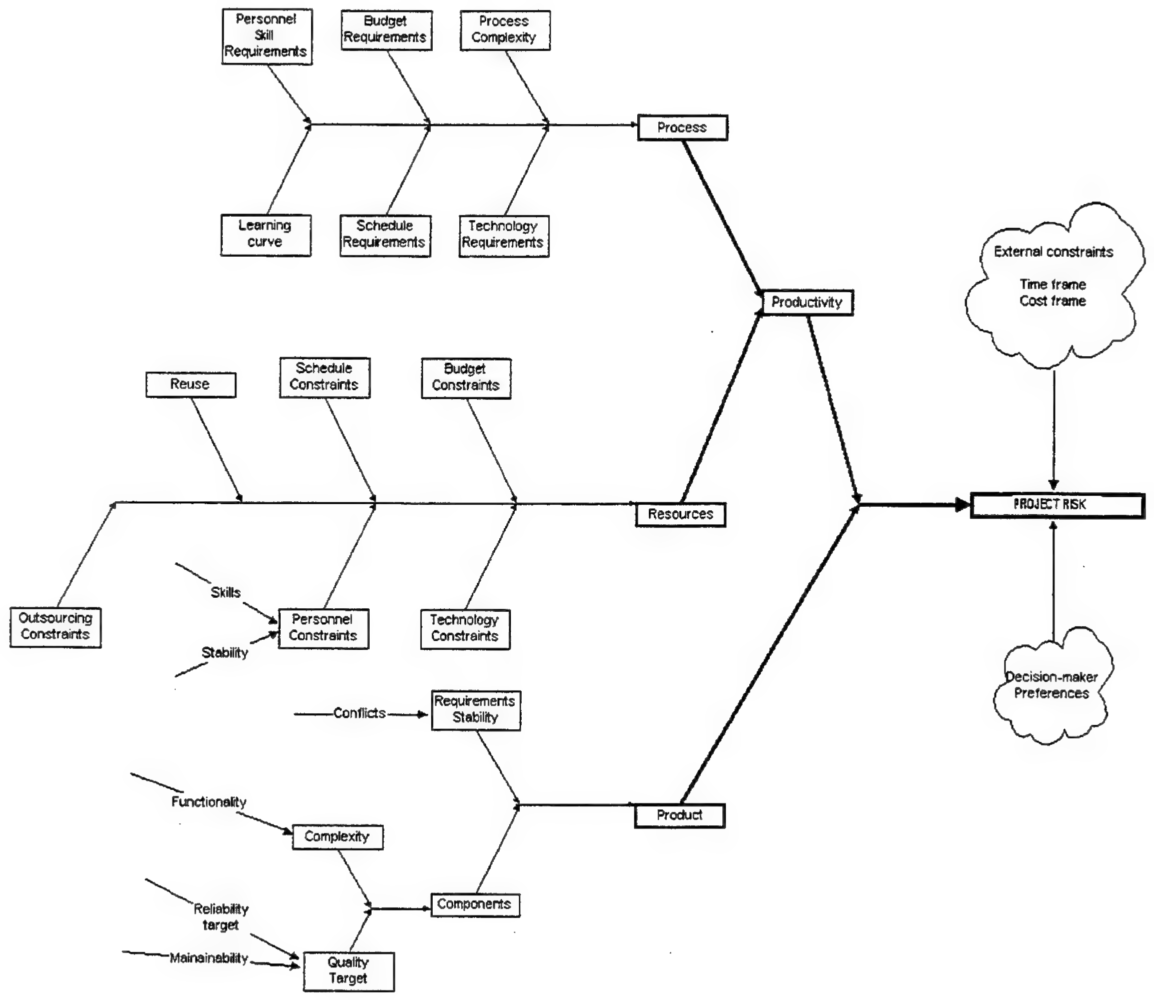


Figure 3.2: Causal Analysis Fishbone Diagram

THIS PAGE INTENTIONALLY LEFT BLANK

IV. RESEARCH DESIGN

As outlined in the introduction, this research is focused primarily upon risk assessment for software engineering. More precisely, it addressed the issue of human dependency in risk assessment of the evolutionary software processes incorporating an automated risk assessment method. Despite the improvements achieved in software processes, software reuse and automated tools, risk assessment for software projects remains as an unstructured problem dependent on human expertise. It is the intention of this research to find ways to transform risk assessment into a structured problem. Solving the risk assessment problem with indicators measured on the early phases constitute the main contribution of this research.

The problem of productivity is partially solved. The industry has enough tools that improve software development productivity. New efforts on this vein are not the solution for the software crisis because the problem in the author's opinion is focused on organizational and human communication issues. Software development is still a human dependent activity requiring lot of human communication, and without appropriate managerial decision support tools, software engineering will remain in its present state. A better understanding of the knowledge about the internal phenomenology of the software life cycle is required to improve software development because it is in the human aspects of the software process where the bottleneck is located now. Without such knowledge, risk assessment is almost impossible.

The primary research question is: What are the early automatically collectable measures from the software process that describe project risk? The risk of the project is related to its probability of success. That is the probability of reaching the objective with the assigned resources in the allocated time. The main point in the question is the discovery of a set of good indicators for risk. These indicators should be recognized during the early phases of the process in order to provide early alert. To answer the research question a literature revision was conducted covering different fields:

1. Review the papers and books about software evolution. This study helped to understand the scope and limitations of such software processes, and it helped to discover the problem.
2. Review of risk management from the operational research point of view. This study provided theoretical background to produce a mathematical model.
3. Review of the literature about risk management in the field of software engineering. This study showed two well-defined groups of researchers. The first group follows a less rigorous and human dependant approach starting from the beginning of the project. This study revealed that this research was original. The second group corresponds to the software reliability field and follows a rigorous approach post mortem. This set of research provided insights of how to link the operational research approaches with the software engineering approaches.
4. Causal analysis was employed to find a set of candidate indicators for risk. The set of candidate indicators was compared with previous research. It was found that requirement variability, personnel turnover and complexity were promising indicators.
5. Review of the software economics research, specially COCOMO and Putnam's models. This study showed that the estimation models available today have some limitations when applied to evolutionary software processes.
6. Experiments to prove the correlation between complexity and size were conducted using the available baselines of projects created by the evolutionary software process, specifically using CAPS.

The second research question is: How can these measures be related in order to assess project risk? Answering this question implies the formalization of a model and its calibration and validation in three ways: a) internal consistency proved by mathematics and statistics; b) black box validation by comparing its outputs in duration and effort with other available models; and c) black box validation against a set of observations. To achieve this last goal a large set of well-measured software projects is required. This set could not be found. A second and more promising alternative was to simulate a set of projects. VitéProject was chosen for the following reasons:

- Availability
- Possibility of customizing
- Includes the model for communications and exceptions
- Given that the proposed model uses parameters collected during the early phases and given that VitéProject requires a complete breakdown structure of the project –that can be done only in the late phases– there exists a considerable time gap between the two measurements. Such time gap is less than conducting a post mortem analysis, but is enough for calibration and validation purposes.

However, the simulation tool is not configured for software projects. To solve this problem it was necessary to review organizational theory and use an expert system (Organizational Consultant) to obtain the correct parameters (see Appendix B). The research ends proposing an extension to the latest version of the graph model, namely relational hypergraph model, to support automated risk assessment.

THIS PAGE INTENTIONALLY LEFT BLANK

V. DEVELOPMENT OF THE MODEL

This chapter applies the framework described in Chapter III, to develop a model for risk identification and risk assessment. First, some concepts about software metrics will be discussed, presenting a small set of metrics in which the model will be based. Next, the model will be discussed with its variables and their relationships.

A. SOFTWARE METRICS

Metrics are a key factor in the identification of threats. Without metrics it is not possible to provide early alerts of risks. There are some erroneous perceptions about metrics that it is necessary to clarify:

- "Metrics act against the creative process." This is an excuse to avoid the use of metrics. Metrics should be collected without the direct intervention of humans. The collection process should be transparent to designers.
- "Metrics represent additional work load." The collection procedure can be automated, so the extra workload is not significant. The analysis of the metrics requires the attention of the project manager, and this is his normal work.
- "The benefits of metrics are unclear." This myth is really irrational. Without measures over the process it is impossible to assess how much effort is required, or what are the risks that should be mitigated.
- "People are afraid of metrics." That is true, and it is very common to find some resistance to the introduction of a metrics plan. It is important to use the metrics to measure the process rather than use them to punish low productivity.

This section describes a set of metrics that support the risk identification strategy. All the metrics presented here are well formed, in the sense that they present the following strengths:

- Robustness. Capacity of being tolerant to variability of the inputs.
- Repeatable. Different observers would arrive to the same measurement no matter how many repetitions.
- Simple. Using the least number of parameters sufficient to obtain an accurate measurement.
- Easy to calculate. They do not require complex algorithms or processes.
- Automatically collected. There is no need of human intervention.

The minimal set of metrics to support the risk assessment model cover three areas: a) requirements, b) personnel (the key resource), and c) complexity. These three groups of metrics correspond to the three risk factors that identified by causal analysis, described in Section 3.1.

1. Metrics for Requirements

a. Birth-rate (BR)

Birth-rate is defined as the percentage of new requirements incorporated in each cycle of the evolution process. This metric shows the explosion of new requirements as a percentage.

BR = (NR / TR) * 100 (%)

where NR = number of new requirements

TR = total number of requirements = PR + NR

PR = previous requirements

b. Death-rate (DR)

Death-rate is defined as the percentage of requirements that are dropped by the customer in each cycle of the evolution process.

DR = (DelR / TR) * 100 (%)

where DelR = number of requirements deleted

TR = total number of requirements (before deletion) = PR + NR

c. ***Change-rate (CR)***

Change-rate is defined as the percentage of requirements changed from the previous version.

CR = (ModR / TR) * 100 (%)

where ModR = number of requirements changed

TR = total number of requirements

From the point of view of the metrics, a change on a requirement can be viewed as a death of the old version and a birth of the new one. This simplification does not imply losses of information about the history of the evolution. The traceability of the evolution remains in the hypergraph model. The simplification just described, enables one to compare birth-rate and death-rate in a bi-dimensional plot that shows four regions: stability region, growing region, volatility region and shrinking region (Fig. 5.1). The graph is double logarithmic, so the borders of the four regions are in the 10% value. Each of these regions has different risk connotations. The arrow shows the normal evolution of the project as the time goes by. During early stages, it is normal for projects being in the growing region. However, if the project continues in this region after many cycles, or return to this region after visiting other regions, something wrong happens. In the first case, this is an indicator that the requirement engineering is not efficient; hence some corrective action should be applied. In the second case, shows evidence of late discovery of some cluster of hidden requirements.

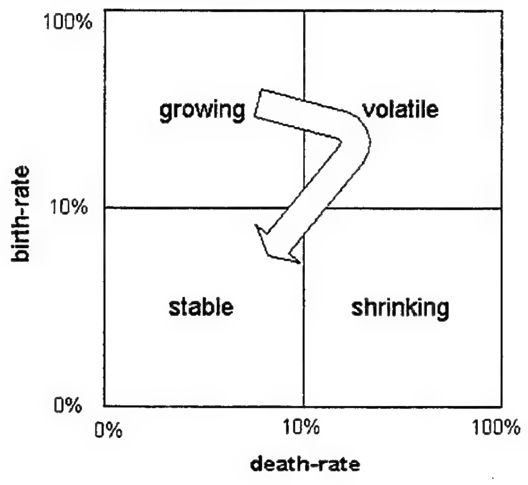


Figure 5.1: Evolution of Requirements

After some cycles, the project should be in the volatile region. If the project does not evolve through the stability region, then there is evidence that the requirements engineering activity is not being efficient and some corrective action is mandatory. It is important to analyze the evolution of the stakeholders' issues and criticisms. It could be also the case that stakeholders have changed their minds. If the project evolves to the shrinking region, and the requirements engineering is working right, there is evidence that the customers are cutting down the project. This can be the indicator of a severe cut in the budget. Finally, any involution to a previous region should be considered as evidence of threats. In such cases a detailed analysis is required to assess the causes of the anomaly. This set of metrics can be collected automatically form the hypergraph and can give early alerts of the threats.

2. Metrics for Personnel

It is necessary to measure the fit between people and their roles in the software process. In order to measure personnel both quantitative and qualitative metrics are required. A skill match between person and job is required to estimate the speed in processing information and rate of exceptions. On the quantitative side it is important to measure the number of people and the turnover. The latter provides information about the expected productivity losses due to training, learning curves and communications. This set of metrics is difficult to collect because people are very reluctant to being measured. The simulations showed that there exists an easier way to measure the productivity fitness observing the ratio between direct working time and idle. Fitness is related to two risk factors: the resources and the process.

3. Metrics for Complexity

In general, the complexity of an object is a function of the relationships among the components of the object. In an early vision of modern object oriented paradigm, (Myers, 1976) introduced three valuable concepts to measure complexity:

- Independence: The independence of each component can reduce the complexity of the system if the components are a partition of the system. So, there is maximum cohesion and minimum coupling.
- Hierarchy: Hierarchical structures allow the stratification of the system in different layers of abstraction.
- Explicit communication: The components should communicate with explicit protocols avoiding any hidden side effects.

Complexity has a direct impact on quality because the likelihood that a component fails is directly related to its complexity. The quality of the product can only be determined at the end of the process. Hence, it is important to measure the complexity as predictor (Munson, 1995). Real time systems present special difficulties in terms of requirement engineering. Some requirements are difficult for the user to provide and for the analysts difficult to determine. The best way to discover these hidden requirements is via prototyping. CAPS is a CASE tool specially suited for this task. It has a graphical easy to understand interface and mapped to a specification language, which in turns generates Ada code. The main components of CAPS are:

- The prototype system description language (PSDL). PSDL is based on data flow under real-time constraints. It uses an enhanced data flow diagram that includes non-procedural control and timing constraints.
- User interface based on a graphic editor with a palette of objects that include operators, inputs, outputs, data flows and operator loops. A browser helps the designer to find reusable components. An expert system provides the capability to generate English descriptions of PSDL specifications.
- The software database system provides the repository for reusable PSDL components.
- The execution support system consists of a translator, scheduling mechanisms and a debugger.

The prototyping process consists of prototype construction and modification (evolution) based on evolving requirements and code generation. Both construction and modification are exploratory activities with a common target: to satisfy multiple users with different and often conflicting points of view. Requirement engineering is a consensus driven activity in which mechanisms for conflict resolution and traceability of requirement evolution represent critical success factors.

The specifications written in PSDL are suitable of being analyzed to compute their complexity. In PSDL code has the following tokens: types, operators, data streams and constraints. Types are declarations of abstract data types required for the system. Operators and data streams are the components of a dataflow graph. Finally, constraints represent the real-time constraints that the system must support.

Two complexity metrics were defined for PSDL: **Fine Granularity Complexity metric (FGC)**, and **Large Granularity Complexity metric (LGC)**. The reason to compute different metrics is because they are indicators of two classes of threats. First, it is necessary to be aware of operators too complex. High complexity on one operator could be caused by poor design and possible can be solved by further decomposition. Second, it is necessary to have a metric to compute the total complexity of the system.

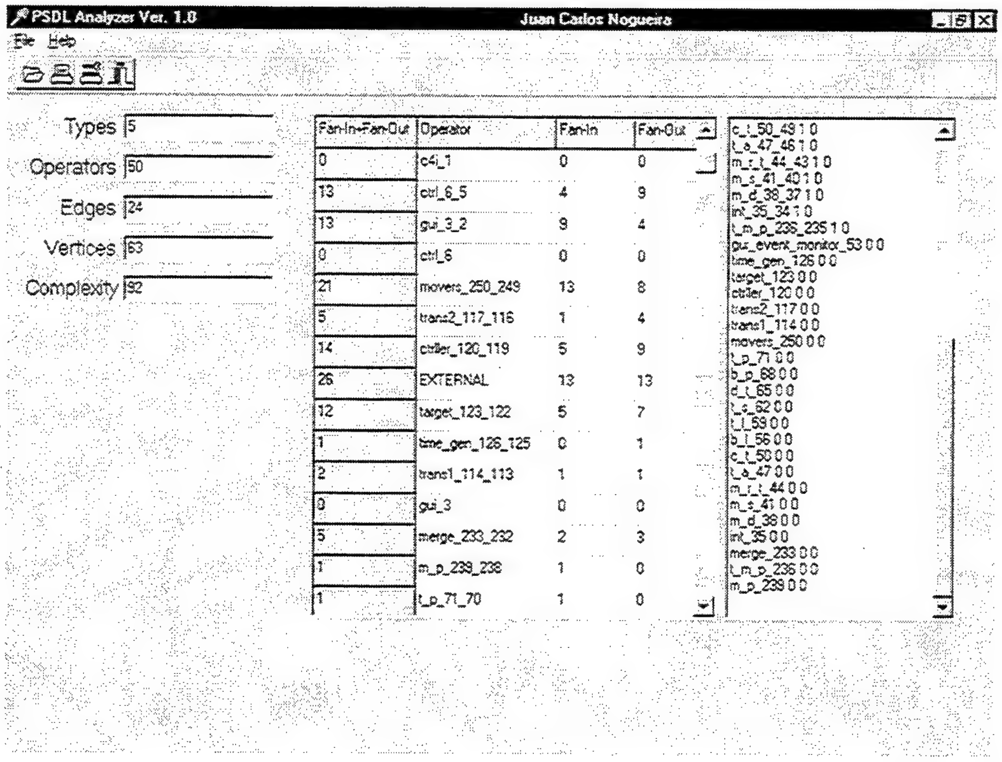
FGC expresses the complexity of each operator in the system and is a function of the fan-in and fan-out data streams related to the operator.

$$\text{FGC} = \text{fan-in} + \text{fan-out}$$

LGC expresses the complexity of the system as a function of the number of operators, data streams, and types.

$$\text{LGC} = \text{O} + \text{D} + \text{T}$$

To analyze PSDL code it was necessary to develop a tool to compute the LGC and FGC. In Figure 5.2 LGC in presented under the title of "Complexity" and FGC is presented under the title "Fan-In+Fan-Out".



Types	5
Operators	50
Edges	24
Vertices	63
Complexity	92

Fan-In+Fan-Out	Operator	Fan-In	Fan-Out
0	c4i_1	0	0
13	ctrl_6_5	4	9
13	gui_3_2	9	4
0	ctrl_6	0	0
21	movers_250_249	13	8
5	trans2_117_116	1	4
14	ctller_120_119	5	9
26	EXTERNAL	13	13
12	target_123_122	5	7
1	time_gen_126_125	0	1
2	trans1_114_113	1	1
0	gui_3	0	0
5	merge_233_232	2	3
1	m_p_239_238	1	0
1	t_p_71_70	1	0

```
c_t_50_49 1 0
t_a_47_46 1 0
m_r_t_44_43 1 0
m_s_41_40 1 0
m_d_38_37 1 0
int_35_34 1 0
t_m_p_236_235 1 0
gui_event_monitor_53 0 0
time_gen_126 0 0
target_123 0 0
ctller_120 0 0
trans2_117 0 0
trans1_114 0 0
movers_250 0 0
t_p_71 0 0
b_p_68 0 0
d_t_65 0 0
t_s_62 0 0
t_l_59 0 0
b_l_56 0 0
c_t_50 0 0
t_a_47 0 0
m_r_t_44 0 0
m_s_41 0 0
m_d_38 0 0
int_35 0 0
merge_233 0 0
t_m_p_236 0 0
m_p_239 0 0
```

Figure 5.2: PSDL Complexity Tool

Figure 5.3 shows the strong correlation between PSDL lines of code and LGC. The correlation coefficient (R) is 0.996.

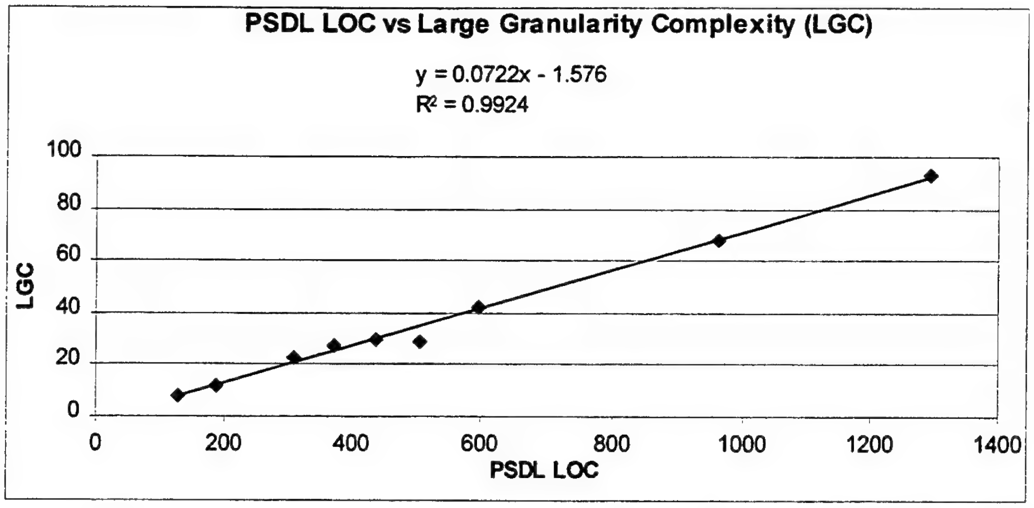


Figure 5.3: Correlation between PSDL and LGC

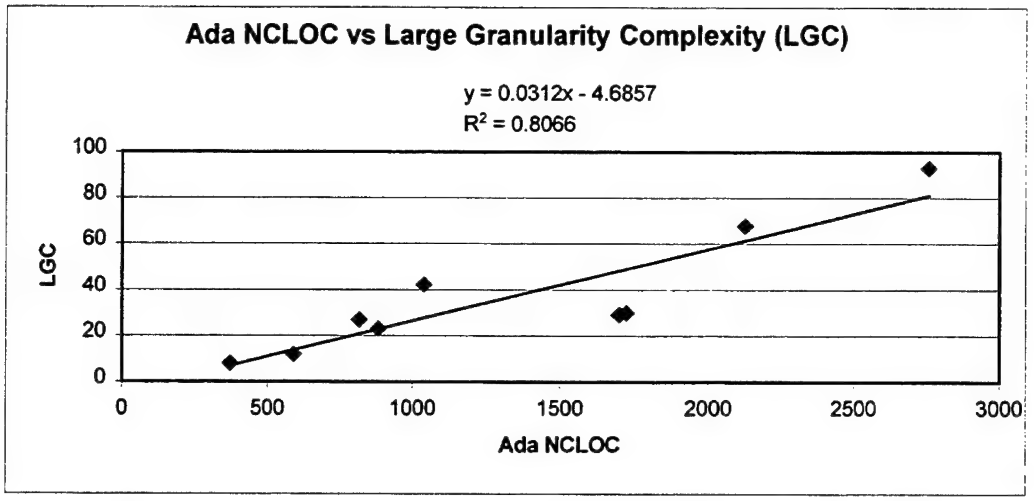


Figure 5.4: Correlation between Ada code and LGC

The comparison between Ada non-comment lines of code of the projects with their complexity measured using LGC shows a strong correlation also (R = 0.898). The complexity metric correlates better with PSDL than with Ada. The reason for this difference is because CAPS automatically generate PSDL. On the other hand, even if CAPS generates part of the Ada code, the designer can add and modify the generated code introducing more variability. Figure 5.4 shows the correlation observed for the same set of projects.

A caveat of this study is that the sample is small, but it includes all the available information at the current time. However, the study suggests the possibility of estimating size in terms of complexity with a useful degree of accuracy.

B. ESTIMATION METHODS

Software projects could be considered as experiments where their cost and schedule are the output measures. It is well known that software projects tend to overrun costs and schedule (this fact has been proved by research and industry) (Boehm, 1981), (Putnam, 1997), (Jones, 1996). There are two possible ways to interpret the result of the experiment. One hypothesis is that this behavior is abnormal, and consequence of lack of process maturity (SEI/CMM approach). Another hypothesis is that this could be a "false-abnormal" behavior assumed abnormal as consequence of inappropriate measurements.

The industry has been using three classes of tools to estimate effort and time that can be applied at different moments during the life cycle, each category being more precise than the previous one but arriving later:

- Very early estimations. This category includes very crude approximations done during the beginning of the process usually by subjective comparisons using previous projects.
- Macro models. This category includes Basic COCOMO, Putnam, Function Points, etc. The estimation is done after completing the requirements phase.
- Micro models. This category includes intermediate and detailed COCOMO, and Pert/CPM/Gantt techniques. The estimation is done after the design when it is possible to have a work breakdown structure. The estimation is the integration of all module estimations.

None of these techniques consider the following characteristics of software projects: a) requirements stability, b) personnel stability, and c) time consumed by communications, exceptions and noise in the process. All the methods use size as input parameter as some kind of derivation from complexity. In many cases the methods to compute such complexities and sizes are questionable. Recently, Stanford University (Levitt, 1999) developed a new generation micro model estimation tool (VitéProject) that addresses some of the previous concerns. However, this tool is useful to control the project but its results arrive too late for early estimation.

How to create a macro model that considers the previous concerns and is able to be used during the early stages of the process? Probabilities can be applied. In 1939 the Swedish physicist Waloddi Weibull introduced a heavy-tailed probability distribution to represent the distribution of the breaking strength of materials (Devore, 1995). There is some controversy about who was the first scientist that introduced this distribution. There is a previous study of 1933 describing the "laws governing the fineness of powdered coal" that used a similar function (Johnson94). Weibull distribution is also known as Weibull-Gnedenko in the Russian literature, and as Frechét for an earlier paper presented in Poland in 1927.

Weibull used this distribution to model strength of Bofors's steel, fiber strength of Indian cotton, length of syrtoideas, fatigue life of steel, statures of adults males, and breadth of beans. The Weibull distribution includes the exponential and the Rayleigh as special cases. It has been used to model different failure rates: a) decreasing (when the shape parameter $\alpha < 1$), b) constant (when $\alpha = 1$ --the exponential case with $\lambda = 1/\beta$--), and c) increasing (when $\alpha > 1$). Many authors (Johnson94, Devore95, Lyu95) advocated the use of this distribution in reliability and quality control. Others like Putnam and Norden used it to model software life cycles. These previous works cited in Chapter II motivated the interest in this distribution.

In some literature (Devore, 1995) and software (Excel), the distribution function is presented with two parameters: α (the shape parameter), and β (the scale parameter that can compress or elongate the curve in the x axis). However, Weibull in his original work mentioned a third parameter, γ, to shift the curve to the right.

A random variable x is said to have a Weibull distribution with parameters α and β (with $\alpha > 0$, $\beta > 0$) if the probability distribution function (pdf) and cumulative distribution function (cdf) of x are respectively:

$$\text{pdf: } f(x; \alpha, \beta) = \begin{cases} 0, & x < 0 \\ (\alpha/\beta^{\alpha})\, x^{\alpha-1} \exp(-(x/\beta)^{\alpha}), & x \geq 0 \end{cases}$$

$$\text{cdf: } F(x; \alpha, \beta) = \begin{cases} 0, & x < 0 \\ 1 - \exp(-(x / \beta)^{\alpha}), & x \geq 0 \end{cases}$$

Lets discuss the meaning of each of the variables in the function:

a) **x** is the random variable under study. In this case, x can be interpreted as development time.

b) α is a shape parameter. It reduces the variability narrowing the shape of the pdf.

c) β is a scale parameter that stretches or compresses the graph in the *x* direction.

d) Note that the functions start at x = 0. A third parameter is required to shift the curves to the right. For that reason was introduced a location parameter γ, which is function of the system complexity. The new functions are then:

$$\text{pdf: } f(x; \gamma, \alpha, \beta) = \begin{cases} 0, & x < \gamma \\ (\alpha/\beta^{\alpha})\, (x - \gamma)^{\alpha-1} \exp(-((x - \gamma)/\beta)^{\alpha}), & x \geq \gamma \quad \text{(Eq. 1)} \end{cases}$$

$$\text{cdf: } F(x; \gamma, \alpha, \beta) = \begin{cases} 0, & x < \gamma \\ 1 - \exp((-(x - \gamma) / \beta)^{\alpha}), & x \geq \gamma \quad \text{(Eq. 2)} \end{cases}$$

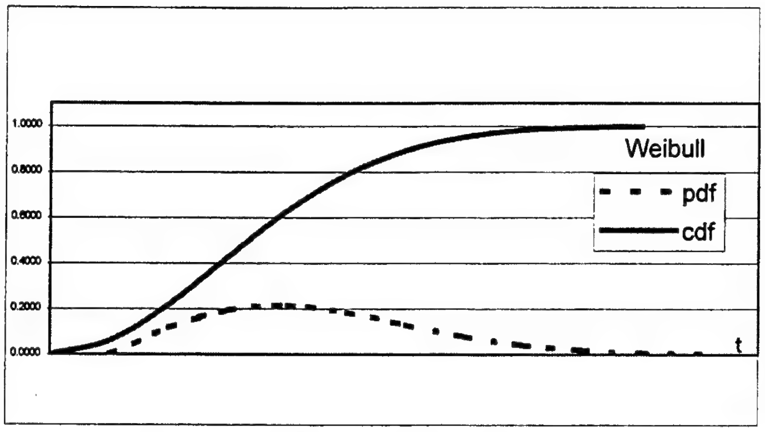


Figure 5.5: Weibull Distribution

C. CONSTRUCTION OF THE MODEL AND SIMULATIONS

1. Finding the Complexity Metric and its Conversion to KLOC

One of the goals of this research was to provide a way to assess the duration of the project given some indicators collected during the requirements phase. In such conditions, code is not available, so the only possible measurements should come from the specification.

Research on Function Points (FP) (Albrecht 1979, 1983) showed that there exists a clear relation between complexity and size in terms of lines of code. However, FP is not well suited for real time systems or object-oriented developments. The reason is that parameters used in FP are not representative of the complexity in such systems. Chapter II discussed in detail this issue. Consequently, it was necessary to look for another way to measure complexity. The observed properties on PSDL showed characteristics that could be used to find the way to calculate complexity. In order to measure the complexity of a module, the count of the fan-in and fan-out is a good estimator. This metric was called Fine Granularity Complexity (FGC). In order to find the complexity of the whole system, the count of PSDL operators (bubbles), data streams (arrows), and types is a good estimator. This metric was called Large Granularity Complexity (LGC).

The observations showed a strong linear correlation between LGC and size of the specification. More interesting was the finding of a strong (but lower) correlation between LGC and the size of the projects in Ada non-comment-lines of code. The size of the project in thousands of non-comment lines of code can be estimated as:

$$KLOC = (32\ LGC + 150) / 1000 \qquad \text{(Eq. 3)}$$

As the complexity grows, the ratio trends to approximately 32 LOC for each unit of LGC. This finding provided us with a method to compute the size of the projects given an early measure of their complexity. This conversion is required to compare how close this approach is with respect to other methods, such as Putnam's and Boehm's, that require size as parameter.

2. Comparison between Putnam's and Boehm's Estimations

Before trying to compare this estimation model with the industries standards (Putnam and COCOMO), an experiment was conducted to compare these two methods (see Chapter II). In the experiment used Basic COCOMO because it is the only one that is a macro model. Intermediate and Detailed COCOMO require a micro calibration that cannot be done until the design is done. The purpose was to analyze early estimations, so Basic COCOMO was the choice. For the comparison Putnam's results were transformed from man-year and years to man-month and months.

The experiment consisted in computing Basic COCOMO and Putnam for fictitious projects from 10 to 1000 KLOC. Basic COCOMO was computed for organic, semidetached, and embedded systems to discriminate between these types of projects. The results showed that in terms of effort, Putnam's method provides an estimation that is close to the average between embedded and semidetached basic COCOMO. In terms of development time, the models are quite similar, Putnam's being more optimistic.

3. Search of the Relationship between Complexity (LGC) and Development Time

Having found a complexity metric suited for this research, the next step was to find the existence of some sort of relationship between LGC and development time. A simple experiment was conducted using the conversion ratio (Eq. 3) to obtain the size inputs for the sample. The sample points were from 1000 LGC to 30000 LGC, which means sample projects from 32 KLOC to almost 1MLOC. The average estimation for the development time using COCOMO and Putnam was computed fro these projects. The sample points are plotted with a smoothing thick line (Fig. 5.6). The logarithmic trendline is plotted as a thin red line. A strong logarithmic correlation ($R^2 = 0.9699$) with the following function was found (Fig. 5.6):

$$\text{Time (months)} = 12.968 \operatorname{Ln}(\text{LGC}) - 82.23 \qquad \text{(Eq. 4)}$$

This equation gives a conservative estimation for projects between 4000 and 20000 LGC (128 and 640 KLOC of Ada). The estimation seems to be too optimistic for projects smaller than 2000 LGC or greater than 25000 LGC.

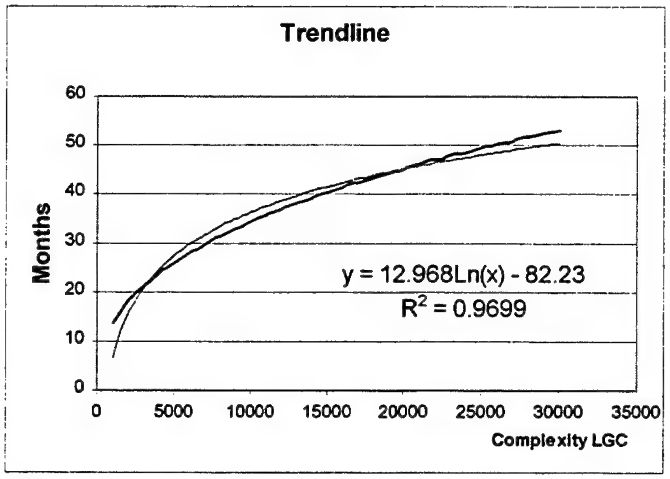


Figure 5.6: Correlation between Development Time and Complexity

4. Search for the Relation between Efficiency and Development Time

Causal analysis showed (Chapter III) that the risk of the project should be dependent on three factors: complexity, productivity and volatility of requirements. The method to compute complexity and the equation to estimate γ, the estimated development time (in months), based on complexity were discussed in the previous sections. Literature in productivity classifies time spent at work into four categories:

- Direct. Time spent working and correcting errors on the product. In VitéProject terminology, it is the sum of work and rework.
- Indirect. Time spent in activities supporting the work such as meetings, coordination, information exchanges, etc. In VitéProject terminology, it is known as coordination time.
- Idle. Time spent without work to do, waiting for some input. In VitéProject terminology, it is known as waiting time.
- Personal. Time spent doing anything except the other categories. VitéProject does not compute this category of time. However, it is loosely related to the noise parameter of the tool.

Examining the time distribution of these categories it is possible to observe a remarkable pattern that differentiates high efficiency scenarios from the low efficiency ones. This effect is independent of the other two variables of the simulation. Hence, this suggests that the time distribution can be a good indicator for the efficiency of the organization The ratio between work and idle time can be automatically captured from the software evolution steps as suggested by [Harn, 1999f]. Figure 5.7 presents the distribution times for the eight scenarios simulated. A pattern of time distributions can be clearly observed. Scenarios with low efficiency have a percentage of idle time greater than 13% of the total development time. The following characteristics can be observed from the simulations:

- Direct work is reduced by 10% when efficiency is high.
- Indirect work is reduced by 40% when the efficiency is high.
- Idle time is reduced by 70% when the efficiency is high.

Low efficiency scenarios can be recognized also by the ratio of the percentage of direct time over percentage of idle time, which was called efficiency ratio (EF):

$$EF = Direct\% / Idle\% \qquad [Eq. 5]$$

For high efficiency scenarios 2.0 < EF, and for low efficient scenarios 0.8 < EF < 2.0.

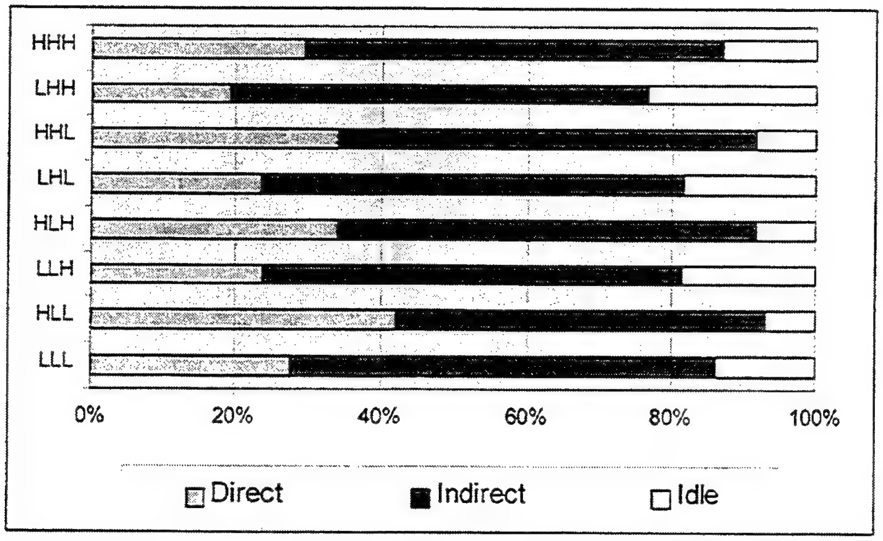


Figure 5.7: Patterns of Time Distribution

The simulations showed that for high efficient scenarios the development time was 60% shorter than for low efficient ones. The reasons why the ratio EF is related to productivity require further study. However, it is possible to conjecture that the reason could be related to:

- Fit of job and people skills.
- People turnover, generating noise and productivity losses derived from training and learning curves.
- Number of people, influencing the productivity in two ways. If the number of people is less than the roles of the software process, then the productivity will be affected because someone will be dividing his attention and effort to more than one role. On the other hand if the number of people exceeds the roles, then the productivity will be affected by additional communications.

5. Search for the Relation between Requirement's Volatility and Development Time

The requirements volatility is obtained by the following formula:

$$RV = INT((BR + DR) / 10) \quad (Eq. 5)$$

For instance if BR = 20% and DR = 10% then RV = 3. The simulations showed a 20% increase on the development time when the requirement's volatility is high (Appendix C).

6. Calibration of the Parameters

To calibrate the values of the parameters described previously, a set of simulations with VitéProject was conducted keeping the values of two variables constant and changing the third one from low to high. The reason to do so, is to isolate the effects of each variable. Having three variables and using two possible values for each one, the universe of scenarios is reduced to the eight (2^3) scenarios showed in Table 5.1:

Table 5.1: Simulated Scenarios

Scenario name	*Productivity*	*Req. volatility*	*Complexity*
LLL	Low	Low	Low
LLH	Low	Low	High
LHL	Low	High	Low
LHH	Low	High	High
HLL	High	Low	Low
HLH	High	Low	High
HHL	High	High	Low
HHH	High	High	High

Each scenario name consists of three letters that correspond to the value of each of the three concepts under study: efficiency ratio (EF), requirements' volatility (RV), and complexity (CX). Each letter could have two values: high (H) or low (L). The simulation tool was configured to run 100 simulations for each scenario, and the organizational parameters were set to match the characteristics of software development.

The simulation reports can be found in Appendix C. Table 5.2 contains the configuration used in the simulation. These values are consequence of an analysis realized with an expert system called Organizational Consultant (Burton, 1998). The tool provides assistance to establish characteristics of organizations. The characteristics of two fictive software development organizations were introduced in the simulations: low efficiency (associated to CMM level 1 or 2), and high efficiency (CMM level 3 or more). The reports of the expert system are presented in Appendix B.

To analyze the effect of efficiency, the results of the simulations of the following scenarios were compared: LLL vs HLL, LLH vs HLH, LHL vs HHL, and LHH vs HHH. It was found that for high productivity scenarios (Hxx) the development time improved in a 60%.

To analyze the effect of requirement volatility, the results of the simulations of the following scenarios were compared: LLL vs LHL, LLH vs LHH, HLL vs HHL, and HLH vs HHH. It was found that high requirement volatility (xHx) degraded the development time in 20 to 30%.

To analyze the effect of complexity, the results of the simulations of the following scenarios were compared: LLL vs LLH, LHL vs LHH, HLL vs HLH, and HHL vs HHH. It was found that high complexity (xxH) degrade the development time in 20 to 30%.

Table 5.2: Configuration Parameters for VitéProject

Scenario	Name	Productivity	eq. Volatility	Complexity	QFD Analysis			Failure	PM		SL		ST		Team
					Req.	Sol.Compl.	Uncert.	Strength	exper.	skill	exper.	skill	exper.	skill	exper.
1	LLL	L	L	L	L	L	L	10	L	L	L	L	L	L	L
3	LLH	L	L	H	M	M	M	10	L	L	L	L	L	L	L
4	LHL	L	H	L	M	M	M	10	L	L	L	L	L	L	L
5	LHH	L	H	H	H	H	H	10	L	L	L	L	L	L	L
6	HLL	H	L	L	L	L	L	10	H	H	H	H	H	H	H
7	HLH	H	L	H	M	M	M	10	H	H	H	H	H	H	H
8	HHL	H	H	L	M	M	M	10	H	H	H	H	H	H	H
9	HHH	H	H	H	H	H	H	10	H	H	H	H	H	H	H

(a) **Parameters of the organization**

- Formalization: Formalization is the degree of informal communications among actors. It was used LOW to indicate likely informal communications.
- Centralization: Centralization is a qualitative degree of decision making and exception handling on lower levels. It was used LOW to indicate that lower levels have decision making power.
- Matrix strength: It was used HIGH to indicate that it is non-departmental work.
- Team experience: It was used LOW or HIGH depending on the scenario.

(b) **Probabilities**

- Functional error rate (0.01 low). Functional errors is the number of generated internal functional errors, showed in the Simulator Analysis Summary.
- Project error rate (0.01 low). Project errors is the number of generated project errors, shown in the Simulator Analysis Summary.
- Information exchange (0.8 high)
- Noise (0.1 normal)

(c) **Communication**

Coordination among activities involves information flow among responsible actors in the project team. A communication represents a packet of information that one actor generates and sends to another actor for processing.

(d) **Coordination**

The Simulator shell Summary Analysis window shows a table that lists a number of costs, durations and quality of each simulated project scenario. Coordination volume is the predicted FTE-time that all actors spend attending to meetings and processing information requests from other actors. The actor FTE sets the number of full-time equivalent (FTE) people that this actor has available to perform activities.

D. THE MODELS

Three models were created with an increasing degree of accuracy. These models are based on:

- Metrics from the three risk factors
- Weibull cumulative density function (Eq. 6)
- The derivation of the time (Eq 7)

The cdf of Weibull is:

$$P(x \leq t) = p = 1 - \exp(-(((t - \gamma) / \beta)^{\alpha})) \qquad \text{(Eq. 6)}$$

$$\therefore 1 - p = \exp(-(((t - \gamma) / \beta)^{\alpha}))$$

$$\therefore \ln(1 - p) = -(((t - \gamma) / \beta)^{\alpha})$$

$$\therefore -\ln(1 - p) = (((t - \gamma) / \beta)^{\alpha})$$

$$\therefore (-\ln(1 - p))^{1/\alpha} = (t - \gamma) / \beta$$

$$\therefore \beta\, (-\ln(1 - p))^{1/\alpha} = t - \gamma$$

$$\therefore \beta\, (-\ln(1 - p))^{1/\alpha} + \gamma = t \qquad \text{(Eq. 7)}$$

Eq. 7 provides the estimated time for a given probability of success p. Note that t and γ should be expressed in the same units. The following notation applies to the algorithms that define the three models:

EF:	efficiency level
RV:	requirements volatility as percentage
CX:	complexity in LGC
γm:	delay in months
γ:	delay in days

All the algorithms can be used to obtain t given EF, RV, CX, and ε (the probability of being correct); or to obtain ε given EF, RV, CX, and t (a given day in the future).

Model 1: This model can be used when the requirements volatility is small.

Algorithm Model 1:

```
i.      If (EF > 2.0)     then begin
                                $\alpha$ = 1.95;
                                $\gamma$m = 0.28 * (13 * ln(LGC) - 82);
                                end
                        else    begin
                                $\alpha$ = 2.5;
                                $\gamma$m = 0.76 * (13 * ln(LGC) - 82);
                                end;
ii.     $\gamma$ = $\gamma$m * 22;         // we assume 22 working days per
        month
iii.    $\beta$ = $\gamma$ / 5.5;
iv.     p = 1 - exp(-(((t - $\gamma$) / $\beta$)$^{\alpha}$));      // P(x<=t)
v.      t = $\beta$ * (-ln(1 - $\varepsilon$)) $^{1/\alpha}$ + $\gamma$;         // time in days
```

Model 2: This model considers the three factors (EF, RV, and CX), but it neglects the combined effect of EF and RV.

Algorithm Model 2:

```
i.      If (EF > 2.0)       then begin
                            $\alpha$ = 1.95;
                            $\gamma$m = 0.28 * (13 * ln(LGC) - 82);
                            end
                            else  begin
                            $\alpha$ = 2.5;
                            $\gamma$m = 0.76 * (13 * ln(LGC) - 82);
                            end;
ii.     $\gamma$ = $\gamma$m * 22;       // we assume 22 working days per month
iii.    If (RV > 30)        then $\beta$ = $\gamma$d / 5.25 // RV more than 30%
                            else $\beta$ = $\gamma$d / 5.9;
iv.     p = 1 - exp(-(((t - $\gamma$) / $\beta$)$^{\alpha}$));      // P(x<=t)
v.      t = $\beta$ * (-ln(1 - $\varepsilon$)) $^{1/\alpha}$ + $\gamma$;         // time in days
```

Model 3: This model considers the three factors as well as the combined effects of EF and RV.

Algorithm Model3:

```
i.    If (EF > 2.0)      then begin
                         $\alpha$ = 1.95;
                         $\gamma m$ = 0.32 * (13 * ln(LGC) - 82);
                         end
                   else begin
                         $\alpha$ = 2.5;
                         $\gamma m$ = 0.85 * (13 * ln(LGC) - 82);
                         end;
ii.   $\gamma$ = $\gamma m$ * 22;       // we assume 22 working days per month
iii.  If (EF > 2.0)      then  $\beta$ = $\gamma$ /(5.71 + (RV - 20) * 0.046)
                         else  $\beta$ = $\gamma$ /(5.47 - (RV - 20) * 0.114);
iv.   p = 1 - exp(-(((t - $\gamma$) / $\beta$)$^\alpha$));       // P(x<=t)
v.    t = $\beta$ * (-ln(1 - $\varepsilon$)) $^{1/\alpha}$ + $\gamma$;         // time in days
```

These three models were tried against 16 simulated projects obtaining the scatter plots of Fig. 5.8, 5.9 and 5.10 respectively. Note the errors as vertical segments between the estimated and real values. The values of R, R2 and standard errors are shown in Table 5.3.

Table 5.3: Accuracy of the Three Models

	Model 1	*Model 2*	*Model 3*
R	0.9867	0.9890	0.9930
R^2	0.9736	0.9781	0.9862
Standard error	30 days	27 days	22 days

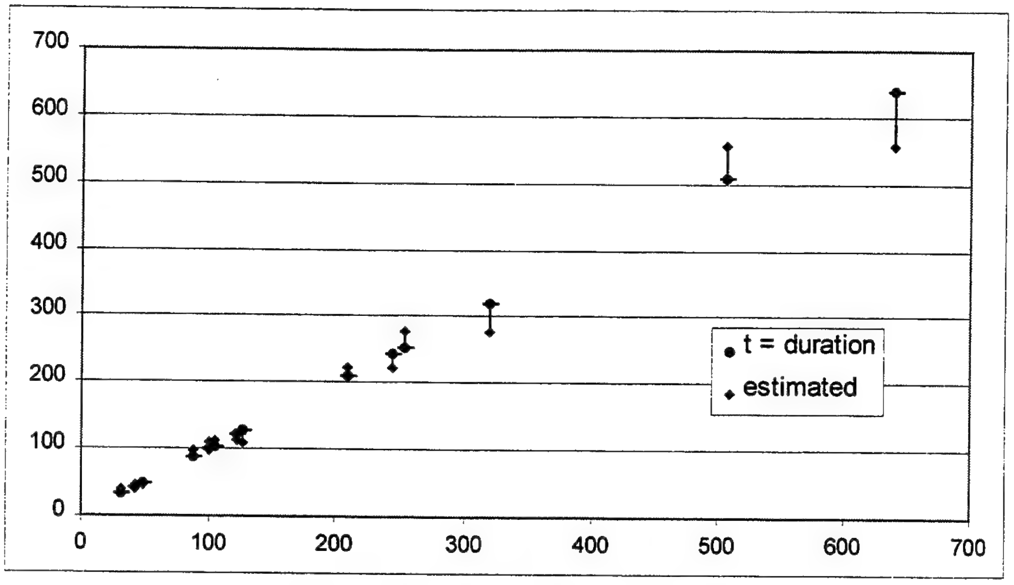


Figure 5.8: Scatter Plot of Model 1

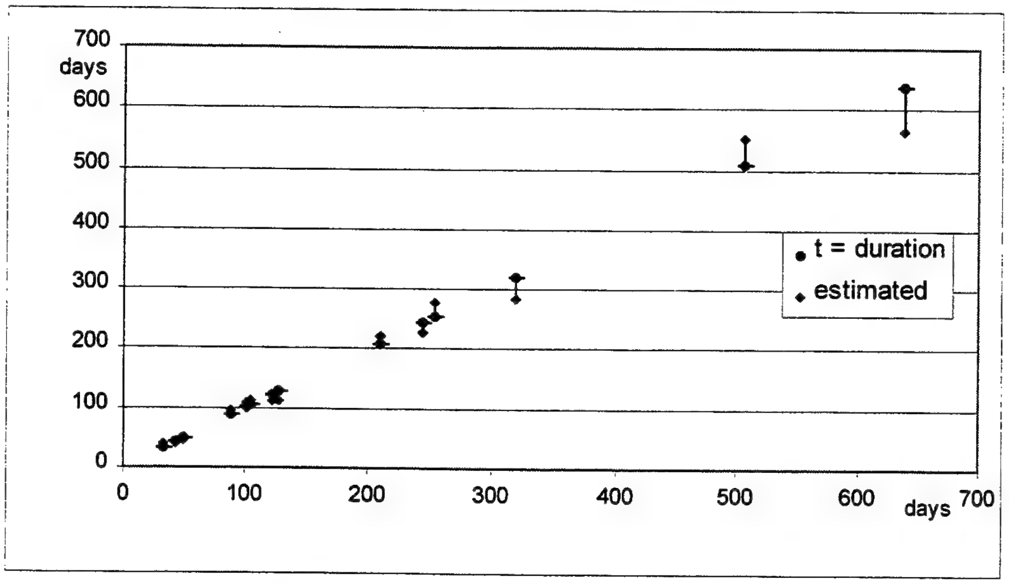


Figure 5.9: Scatter Plot of Model 2

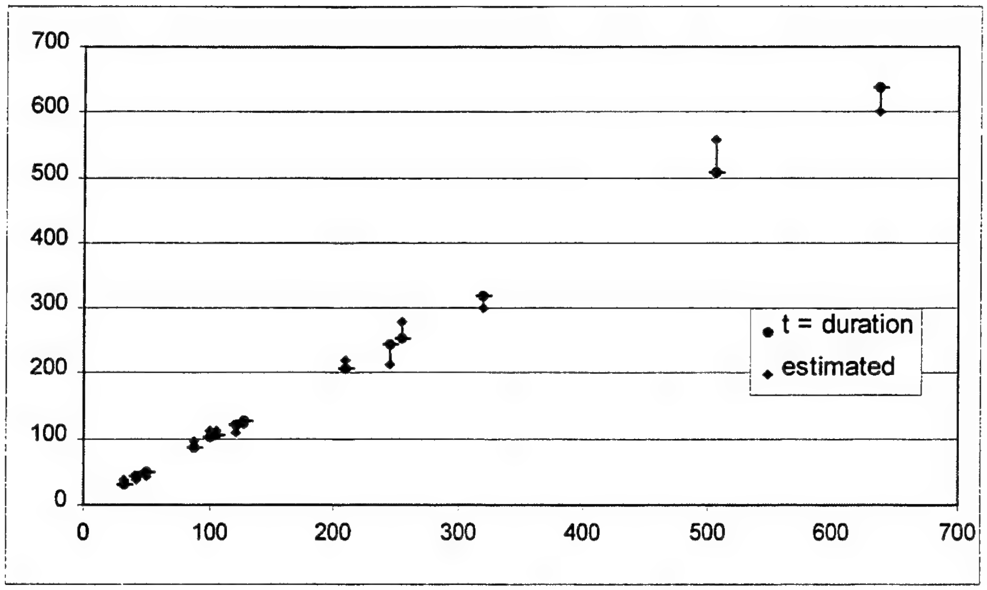


Figure 5.10: Scatter Plot of Model 3

E. INTEGRATION WITH THE EVOLUTIONARY SOFTWARE PROCESS

The evolutionary prototyping software process (Fig. 5.11) is a directed graph with two cycles. Initially, the analysts collect a set of issues, which represent concerns and preliminary goals of the customers, and transform them into a more elaborated level of description called requirements using a requirements analysis step.

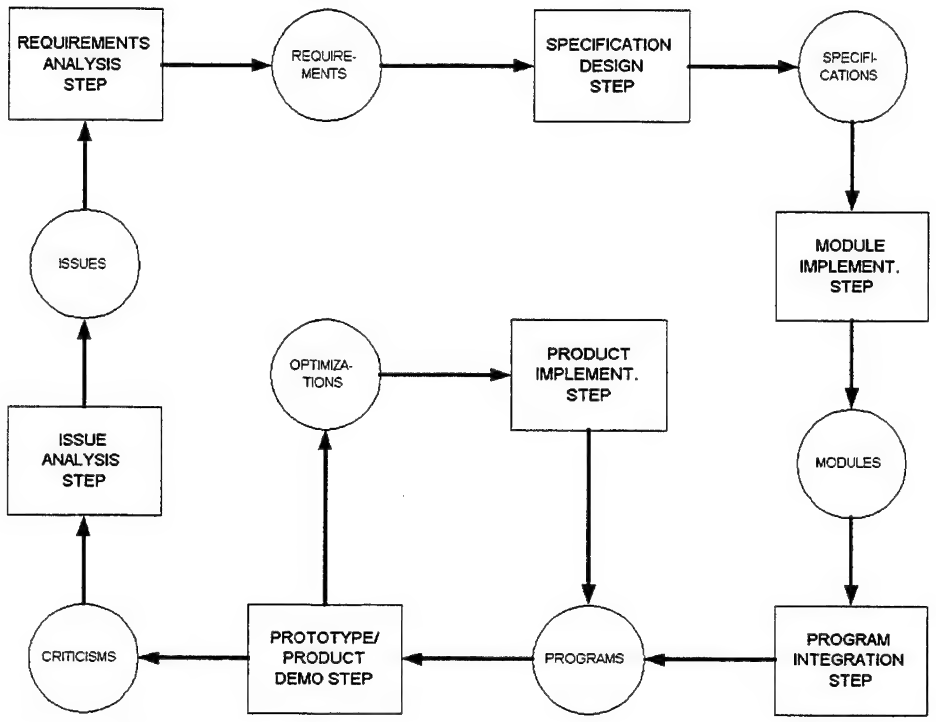


Figure 5.11: The Evolutionary Prototyping Software Process. The vertices in the graph are represented by rectangles. The arcs labeled with circles represent the edges of the digraph.

The requirements are transformed into specifications, probably in PSDL, during the specification design step. In the module implementation step the specifications are automatically converted into code using an appropriate CASE tool such CAPS. The program integration step transforms the modules obtained by the generator are integrated into a program, possibly adding code created by programmers and reusable components. This step includes integration testing and debugging. The program is demonstrated to the customer in a prototype demo step that has two possible outcomes: a) the customer is not satisfied and introduces criticisms, or b) the product conforms the needs and expectations of the customer. In the first case, the process continues by analyzing the criticisms during an issue analysis step that produces new issues closing the external cycle in the graph. In the second case, the prototype contains all the required functionality, so a set of optimizations is introduced during a product implementation step. The resulting product is presented again to the customer during a product demo step closing the internal cycle of the graph.

It is required the introduction of a new vertex in the graph to contain the risk assessment step. A risk assessment step can be automatically done after the completion of the specifications. From the specifications it is feasible to derive the complexity of the product. This information is used together with personnel and organizational information, and with metrics of requirements collected from the baselines, to produce the risk assessment. The risk assessment step integrates these measures with issues in the issue analysis steps (Fig. 5.12).

The development life cycle can be visualized a succession of prototyping developments with increasing functionality followed by a final optimization that produces the system. Each of these phases has the same activity pattern, so its reasonable to suppose that the delivery time for each one has a probability distribution from the Weibull but with different parameters.

During each phase a certain number of problem events occur. A problem event is an effort-consuming situation that introduces a certain amount of functional complexity to be solved (caused by a new requirement, a change on a requirement, or as the consequence of rework), and a certain amount of information exchange.

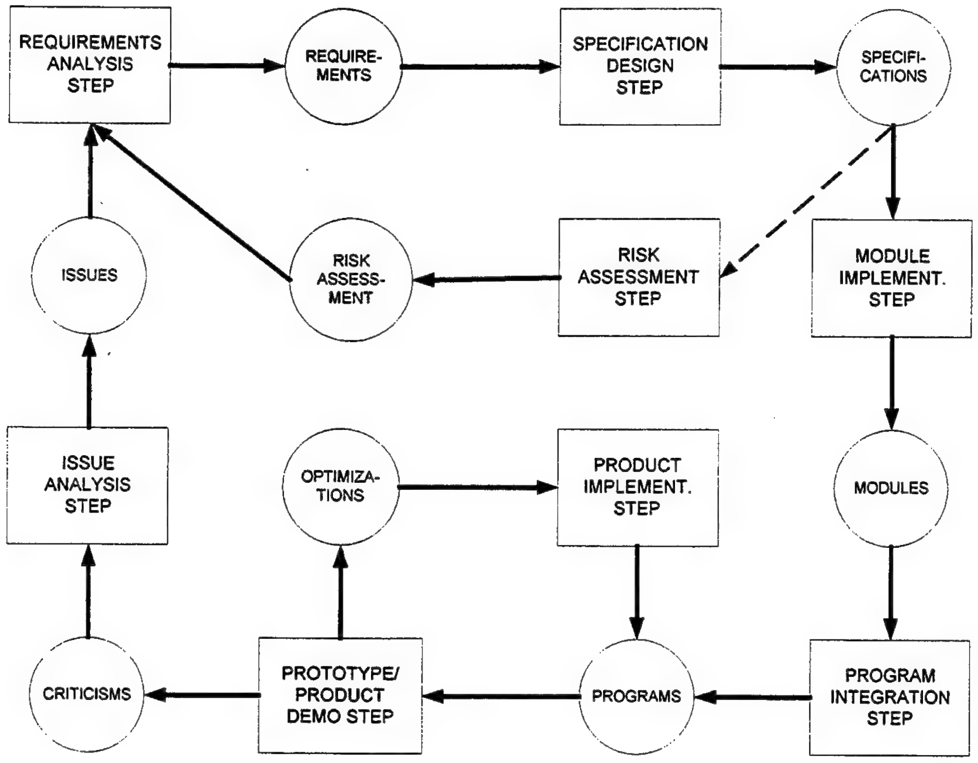


Figure 5.12 The Proposed Improvement

It is supposed that the occurrence of problem events in each phase follows a Poisson distribution with different mean (λ) for each phase. So, the entire development life cycle is a non-homogeneous Poisson process (Fig. 5.13). The assumption of this distribution is based on the following reasoning:

- There exists a certain rate of occurrence of events.
- The probability of more than one event occurring in a time interval depends on the length of the interval.
- The number of events during one time interval is independent of the number received prior this time interval.

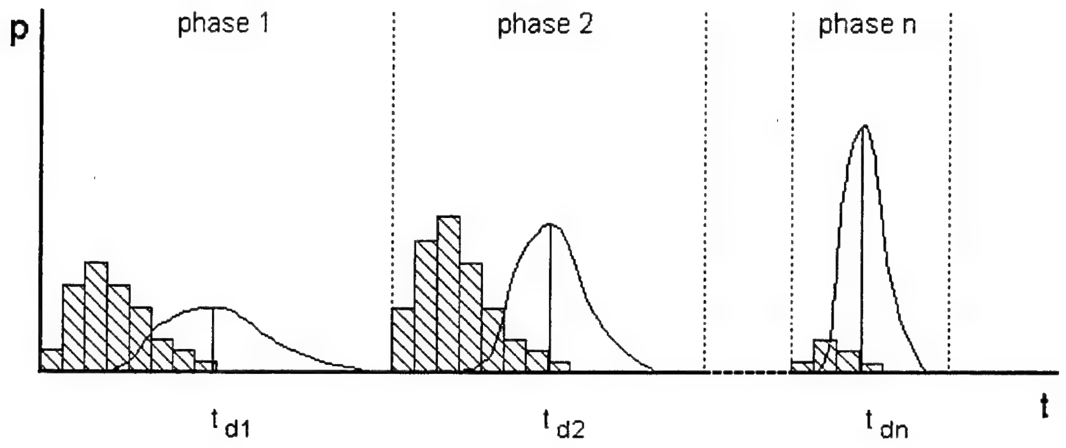


Figure 5.13: The Development Life Cycle. The shadow represents the non-homogeneous Poisson process of the problem events. The curves represent the Weibull probability distributions for the development time of each phase.

VI. CONCLUSIONS

This thesis introduced a formal model to assess the risk of software projects based on metrics automatically collectable from the project baseline. The model enables a project manager to evaluate the probability of success of the project very early in the life cycle. The problem of subjectivity in risk assessment is addressed by using a formal method. Any decision-maker will arrive to the same estimations, independently of his expertise.

A second benefit of this approach is that the model is an estimation tool for time and effort, which improves the state of the art. The model addresses the weaknesses of current standards for estimation because the constraint of frozen requirements, existent in COCOMO 81, COCOMO II and Putnam, is not an issue in this model.

Finally, the research has been addressed using simulations and a small set of real projects. It is necessary to conduct a survey with a large set of real projects to confirm the results.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A
FORMAL DEFINITION OF THE RELATIONAL HYPERGRAPH MODEL

Definition 1: Directed hypergraph (Harn, 1999f). A *directed hypergraph* is a tuple H = (N, E, I, O) where N is a set of nodes, E is a set of hyperedges, I is a function giving the set of input nodes of each hyperedge, and O is a function giving the output nodes of each hyperedge.

Definition 2: Path (Harn, 1999f). A *path* p from node n_1 to node n_k is a sequence of hyperedges $e_1, \ldots, e_{k-1}$ ($k>0$), and a sequence of nodes $n_1, \ldots, n_k$.

Definition 3: Acyclic hypergraph (Harn, 1999f). A hypergraph H = (N, E, I, O) is *acyclic* if and only if there is no path from any node in H to itself.

Definition 4: Reachable (Harn, 1999f). A set N of nodes is *reachable* from a set R of nodes if and only if there is path to each node $n \in N$ from some node $r \in R$. A hypergraph H, is reachable from a set R of its nodes, if and only if all its nodes are reachable from R. The *root* of the hypergraph H is a node from which H is reachable. A *leaf* of H is a node from which no other node is reachable.

Definition 5: Composite node and composite edge (Harn, 1999f). A composite node is a set of nodes, and a composite edge is a set of edges.

Definition 6: Hypergraph set (Harn, 1999f). The hypergraph set is the union of nodes and edges of a set of hypergraphs.

Definition 7: Minimal hypergraph (Harn, 1999f). Let N_{in} and N_{out} be input and output nodes of a hyperedge e in the hypergraph H = (N, E, I, O). A *minimal hypergraph* H_m = ($N_{in} \cup N_{out}$, {e}, I, O) is a minimal unit of the hypergraph whose edge set has only one edge e, and where N_{in} = I(e) and N_{out} = O(e).

Definition 8: Refinement of a composite node (Harn, 1999f). Let H = (N, E, I, O). The *refinement of a composite node* n $\in$ N is a directed minimal hypergraph H_m = ($N_{in} \cup N_{out}$, {e}, I, O), where the input node set N_{in} = {n1, …, nn}, the output node set N_{out} = {n}, and the edge set is {e}. The edge e is *called decomposition edge* and relates the refinement node with its decomposition.

Definition 9: Opposite hypergraph (Harn, 1999f). Let H = (N, E, I, O) then its *opposite hypergraph* H^{op} = (N, E, O, I).

Definition 10: Hyperpath (Harn, 1999f). A *hyperpath* in the hypergraph H = (N, E, I, O), is the minimal hypergraph from a set of nodes N_1 to another set of nodes N_2 where $N_1 \subseteq$ N and $N_2 \subseteq$ N.

Definition 11: Refinement of a composite edge (Harn, 1999f). Let H = (N, E, I, O). *The refinement of a composite edge* e = e_1, …, e_n where e $\in$ E, is a hypergraph set of minimal hypergraphs R = ($N_{in} \cup N_{out}$, e, I, O). N_{in} = I(e), N_{out} = O(e), and e_1, …, e_n are called *subedges*.

Definition 12: Refinement of a minimal hypergraph (Harn, 1999f). Let $H_m = (N_{in} \cup N_{out}, \{e\}, I, O)$ be a minimal hypergraph. The refinement of a minimal hypergraph is a hypergraph set $R = H_{in} \cup H_{out} \cup H_e$, where H_{in} is a refinement of N_{in}, H_{out} is a refinement of N_{out}, and H_e is a refinement of e. H_m can be viewed as a graph composed by two nodes (N_{in}, N_{out}) and one edge (e) where N_{in} and N_{out} are hypergraphs and e is hyperedge.

Definition 13: Evolutionary hypergraph (Harn, 1999f). An *evolutionary hypergraph* is a labeled, directed, and acyclic hypergraph H = (N, E, I, O) together with label functions that give *component attributes* to the nodes and *step attributes* to the edges.

Definition 14: Top-level evolution step (Harn, 1999f). A hyperedge is called *top-level evolution step* if there are no parent evolution steps.

Definition 15: Atomic evolution step (Harn, 1999f). An *atomic evolution step* is an atomic (non decomposable) edge.

Definition 16: Top-level evolutionary hypergraph (Harn, 1999f). A *top-level evolutionary hypergraph* is an evolutionary hypergraph which its edges are top-level evolution steps.

Definition 17: Atomic evolutionary hypergraph (Harn, 1999f). An *atomic evolutionary hypergraph* is an evolutionary hypergraph with an atomic evolution step as its hyperedge.

Definition 18: Primary input (Harn, 1999f). *Primary inputs* are different versions of the output component of an evolutionary step.

Definition 19: Secondary inputs (Harn, 1999f). *Secondary inputs* are all other input components required in an evolutionary step that are not primary inputs.

Definition 20: Primary-input-driven hypergraph (Harn, 1999f). An evolutionary hypergraph is called *primary-input-driven* if and only if its nodes are primary inputs.

Definition 21: Secondary-input-driven hypergraph (Harn, 1999f). An evolutionary hypergraph is called *secondary-input-driven* if and only if its nodes are secondary inputs.

Definition 22: Relational hypergraph (Harn, 1999f). A *relational hypergraph* is an evolutionary hypergraph in which the dependency relationships between components and steps can have a hierarchy of specialized interpretations.

Definition 23: Software prototyping demo step (Harn, 1999f). A *software prototyping demo step* is a step in which the input components are a set of criticisms (C1), a set of programs (P), a set test scenarios (T), and a set of stakeholders (U), producing an output component set of criticisms (C2).

Definition 24: Issue analysis step (Harn, 1999f). A *issue analysis step* is a step in which the input components are a set of previous issues (J1), a set of stakeholders (U), a set of criticisms (C), producing an output component set of new issues (J2).

Definition 25: Requirement analysis step (Harn, 1999f). A *requirement analysis step* is a step in which the input components are a set of previous requirements (R1), a set of issues (J), a set of stakeholders (U), producing an output component set of new requirements (R2).

Definition 26: Specification design step (Harn, 1999f). A *specification design step* is a step in which the input components are a set of previous specifications (S1), a set of stakeholders (U), a set of requirements (R), producing an output component set of new specifications (S2).

Definition 27: Module implementation step (Harn, 1999f). A *module implementation step* is a step in which the input components are a set of previous modules (M1), a set of stakeholders (U), a set of specifications (S), producing an output component set of new modules (M2).

Definition 28: Program integration step (Harn, 1999f). A *program integration step* is a step in which the input components are a set of previous programs (P1), a set of stakeholders (U), a set of modules (M), producing an output component set of new programs (P2).

Definition 29: Software product demo step (Harn, 1999f). A *software product demo step* is a step in which the input components are a set of previous optimizations (K1), a set of stakeholders (U), a set of programs (P), a set of test scenarios (T), producing an output component set of new optimizations (K2).

Definition 30: Software product implementation step (Harn, 1999f). A *software product implementation step* is a step in which the input components are a set of previous versions of programs (P1), a set of stakeholders (U), a set of optimizations (K), producing an output component set of new programs (P2).

Definition 31: Software prototyping evolution process (Harn, 1999f). A *software prototyping evolution step* is a hypergraph with a path with the following properties:

(1) Steps are software prototype or product demo, issue analysis, requirement analysis, specification design, module implementation and program integration.

(2) Nodes are old version programs, criticisms, issues, requirements, specifications, modules, and new version programs.

Definition 32: Software product generation process (Harn, 1999f). A *software product process* is a relational hypergraph with a path with the following properties:

(1) Steps are software prototype or product demo, and program integration.

(2) Nodes are new version prototypes or old version programs, optimizations, and new version programs.

Definition 33: Software evolution process (Harn, 1999f). A *software evolution process* is a relational hypergraph with a combined structure of software prototyping evolution processes and software product generation processes.

Definition 34: Top-level relational hypergraph net (Harn, 1999f). A *top-level relational hypergraph* is a set composed by a set of primary inputs, one or more sets of secondary inputs, and a set of output nodes to a top-level evolution step. (Harn, 1999f) called this is concept SPIDER (Step Processed in Different Entrance Relationships).

Definition 35: Atomic relational hypergraph net (Harn, 1999f). An *atomic relational hypergraph* is a set composed by a set of primary inputs, one or more sets of secondary inputs, and a set of output nodes to an atomic evolution step. (Harn, 1999f) called this is concept atomic SPIDER.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B

ANALYSIS WITH ORGANIZATIONAL CONSULTANT

The following reports were produced using Organizational Consultant expert system. The first report analyze a fictive organization "Software Engineering" which represents a typical public software development department below CMM level 3. The second report analyzes the same organization after reaching CMM level 3.

REPORT SUMMARY - Software Engineering

Time: 2:17:22 PM, 12/29/99
Scenario: Scenario 1

INPUT DATA SUMMARY

The description below summarizes and interprets your answers to the questions about your organization and its situation. It states your answers concerning the organization's current configuration, complexity, formalization, and centralization. Your responses to the various questions on the contingencies of age, size, technology, environment, management style, cultural climate and strategy factors are also given. The writeup below summarizes the input data for the analysis.

- Software Engineering has an adhocracy configuration (cf 100).
- Software Engineering has a small number of different jobs (cf 100).
- Of the employees at Software Engineering 76 to 100 % have an advanced degree or many years of special training (cf 100).
- Software Engineering has 3 to 5 vertical levels separating top management from the bottom level of the organization (cf 100).
- The mean number of vertical levels is 3 to 5 (cf 100).
- Software Engineering has 1 or 2 separate geographic locations (cf 100).
- Software Engineering's average distance of these separate units from the organization's headquarters is of no relevance because there is only one site undetermined (cf 100).
- An undetermined number of Software Engineering's total workforce is located at these separate units (cf 100).
- Job descriptions are available for none or an undetermined number of employees (cf 100).
- Where written job descriptions exist, the employees are supervised an undetermined manner to ensure compliance with standards set in the job description (cf 100).
- The employees are allowed to deviate in an undetermined way from the standards (cf 100).
- 0 to 20 % non-managerial employees are given written operating instructions or procedures for their job (cf 100).
- The written instructions or procedures given are of no relevance as there are no written instructions or they may be undetermined (cf 100).
- Supervisors and middle managers are to some extent free from rules, procedures, and policies when they make decisions (cf 100).
- Less than 20 % of all the rules and procedures that exist within the organization are in writing (cf 100).
- Top Management is to a great extent involved in gathering the information they will use in making decisions (cf 100).
- Top management participates in the interpretation of 61 to 80 % of the information input (cf 100).
- Top management directly controls 21 to 40 % of the decisions executed (cf 100).
- The typical middle manager has little discretion over establishing his or her budget (cf 100).
- The typical middle manager has little discretion over how his/her unit will be evaluated (cf 100).
- The typical middle manager has little discretion over the hiring and firing of personnel (cf 100).

- The typical middle manager has little discretion over personnel rewards - (ie, salary increases and promotions) (cf 100).
- The typical middle manager has some discretion over purchasing equipment and supplies (cf 100).
- The typical middle manager has little discretion over establishing a new project or program (cf 100).
- The typical middle manager has very great discretion over how work exceptions are to be handled (cf 100).
- Software Engineering has 25 employees (cf 100).
- Software Engineering's age is young (cf 100).
- Software Engineering's ownership status is public (cf 100).
- Software Engineering has some different products (cf 100).
- Software Engineering has few different markets (cf 100).
- Software Engineering only operates in one country (cf 100).
- Software Engineering has no different products in the foreign markets (cf 100).
- Software Engineering's major activity is categorized as service (cf 100).
- Software Engineering has a specialized customer-oriented service technology (cf 100).
- Software Engineering has undetermined technology (cf 100).
- Software Engineering's technology is undetermined with respect to divisibility (cf 100).
- Software Engineering's technology dominance is strong (cf 100).
- Software Engineering has given no information about a possible advanced information system (cf 100).
- Software Engineering's environment is complex (cf 100).
- The uncertainty of Software Engineering's environment is high (cf 100).
- The equivocality of the organization's environment is high (cf 100).
- Software Engineering's environment has an undetermined level of hostility (cf 100).
- Top management prefers to make resource allocations and detailed operating decisions (cf 100).
- Top management primarily prefers to make long-term decisions (cf 100).
- Top management has a preference for very aggregate information when making decisions (cf 100).
- Top management has a preference for some proactive actions and some reactive actions (cf 100).
- Top management is risk averse (cf 100).
- Top management has a preference for a combination of motivation and control (cf 100).
- Software Engineering operates in an industry with a medium capital requirement (cf 100).
- Software Engineering has a high product innovation (cf 100).
- Software Engineering has a high process innovation (cf 100).
- Software Engineering has a high concern for quality (cf 100).
- Software Engineering's price level is undetermined relative to its competitors (cf 100).
- The level of trust is high (cf 100).
- The level of conflict is low (cf 100).
- The employee morale is not known (cf 100).
- Rewards are given in a not known fashion (cf 100).
- The resistance to change is not known (cf 100).
- The leader credibility is high (cf 100).
- The level of scapegoating is low (cf 100).

THE SIZE

The size of the organization - large, medium, or small - is based upon the number of employees, adjusted for their level of education or technical skills.
Based on the answers you provided, it is most likely that your organization's size is medium (cf 50).
More than 75 % of the people employed by Software Engineering have a high level of education. Adjustments are made to this effect. The adjusted number of employees is lower than 500 but greater than 100 and Software Engineering is categorized as medium. However, for this adjusted number this size does not have a major effect on the organizational structure.

THE CLIMATE

The organizational climate effect is the summary measure of people and behavior.

Based on the answers you provided, it is most likely that the organizational climate is a group climate (cf 76).
It could also be that climate is a developmental (cf 73).
The group climate is characterized as a friendly place to work where people share a lot of themselves. It is like an extended family. The leaders, or head of the organization, are considered to be mentors and, perhaps even parent figures. The organization is held together by loyalty or tradition. Commitment is high. The organization emphasizes the long-term benefit of human resource development with high cohesion and morale being important. Success is defined in terms of sensitivity to customers and concern for people. The organization places a premium on teamwork, participation, and consensus.
When the organization has a high level of trust it is likely that the organization has a group climate. An organization with little conflict can be categorized to have group climate. High leader credibility characterizes an organization with a group climate. An organization with a low level of scapegoating may have a group climate.
The developmental climate is characterized as a dynamic, entrepreneurial and creative place to work. People stick their necks out and take risks. The leaders are considered to be innovators and risk takers. The glue that holds organizations together is commitment to experimentation and innovation. The emphasis is on being on the leading edge. Readiness for change and meeting new challenges are important. The organization's long-term emphasis is on growth and acquiring new resources. Success means having unique and new products or services and being a product or service leader is important. The organization encourages individual initiative and freedom.
When the organization has a high to medium level of trust it is likely that the organization has a developmental climate. An organization with low level of conflict can be categorized to have a developmental climate. Medium to high leader credibility characterizes an organization with a developmental climate. An organization with a medium level of scapegoating may have a developmental climate.

THE MANAGEMENT STYLE

The level of management's microinvolvement in decision making is the summary measure of management style. Leaders have a low preference for microinvolvement; managers have a high preference for microinvolvement.
Based on the answers you provided, it is most likely that your management profile has a medium preference for microinvolvement (cf 78).
The management of Software Engineering has a preference for letting some decisions be made by other managers. This will lead toward a medium preference for microinvolvement. The management of Software Engineering has a preference for taking actions on some decisions and being reactive toward others. This will lead toward a medium preference for microinvolvement. Management has a preference for using both motivation and control to coordinate the activities, which leads toward a medium preference for microinvolvement.

THE STRATEGY

The organization's strategy is categorized as one of either prospector, analyzer with innovation, analyzer without innovation, defender, or reactor. These categories follow Miles and Snow's typology. Based on your answers, the organization has been assigned to a strategy category. This is a statement of the current strategy; it is not an analysis of what is the best or preferred strategy for the organization.
Based on the answers you provided, it is most likely that your organization's strategy is an analyzer with innovation strategy (cf 68).
It could also be: a prospector (cf 64).
An organization with an analyzer with innovation strategy is an organization that combines the strategy of the defender and the prospector. It moves into the production of a new product or enters a new market after viability has been shown. But in contrast to an analyzer without innovation, it has innovations that run concurrently with the regular production. It has a dual technology core.
An organization with a medium capital investment is likely to have some capabilities rather fixed, but can also adjust. The analyzer with innovation which seeks new opportunities but also maintains its profitable position is appropriate. With a concern for high quality an analyzer with innovation strategy is a likely

strategy for Software Engineering. With top management preferring a medium level of microinvolvement top management wants some influence. This can be obtained via control over current operations. Product innovation should be less controlled. The strategy is therefore likely to be analyzer with innovation.
An organization with a prospector strategy is an organization that continually searches for market opportunities and regularly experiments with potential responses to emerging environmental trends. Thus, the organization is often the creator of change and uncertainty to which its competitors must respond. However, because of its strong concern for product and market innovation, a prospector usually is not completely efficient.
With a concern for high quality a prospector strategy is a likely strategy for Software Engineering.

THE CURRENT ORGANIZATIONAL CHARACTERISTICS

Based on your answers, the organization's complexity, formalization, and centralization have been calculated. This is the current organization. Later in this report, there will be recommendations for the organization.
The current organizational complexity is medium (cf 100).
The current horizontal differentiation is medium (cf 100).
The current vertical differentiation is low (cf 100).
The current spatial differentiation is low (cf 100).
The current centralization is medium (cf 100).
The current formalization is low (cf 100).
The current organization has been categorized with respect to formalization, centralization, and complexity. The categorization is based on the input you gave and does not take missing information into account.

SITUATION MISFITS

A situation misfit is an unbalanced situation among the contingency factors of management style, size, environment, technology, climate, and strategy.
The following misfits are present: (cf 100).
Software Engineering has both an analyzer strategy and few products. Generally, more products are required for an analyzer. A few products may be reasonable in the short run, but an analyzer should be in constant consideration of new possibilities. When a few, unchanging products become the norm, the analyzer should broaden its scope of new opportunities.

ORGANIZATIONAL CONSULTANT RECOMMENDATIONS

Based on your answers about the organization, its situation, and the conclusions with the greatest certainty factor from the analyses above Organizational Consultant has derived recommendations for the organization's configuration, complexity, formalization, and centralization. There are also recommendations for coordination and control, the appropriate media richness for communications, and incentives. More detailed recommendations for possible changes in the current organization are also provided.

ORGANIZATIONAL CONFIGURATIONS

The most likely configuration that best fits the situation has been estimated to be a matrix configuration (cf 59).
A matrix structure is a structure that assigns specialists from functional departments to work on one or more interdisciplinary teams that are led by project leaders. Permanent product teams are also possible. A dual hierarchy manages the same activities and individuals at the same time.
When Software Engineering's environment has neither low equivocality nor low complexity, the configuration should be matrix. When Software Engineering is of medium size, the configuration can be a matrix configuration. The matrix configuration is a more likely configuration when Software Engineering has a unit production technology.

ORGANIZATIONAL CHARACTERISTICS

The recommended degree of organizational complexity is medium (cf 43).
Medium size organizations should have medium organizational complexity. Top management of Software Engineering has a preference for a medium level of microinvolvement, which drives the organizational complexity towards medium. A group climate in the organization requires a medium level of complexity with a low level of vertical differentiation.
The recommended degree of horizontal differentiation is low (cf 28).
It, too, could be: medium (cf 19).
The recommended degree of vertical differentiation is low (cf 38).
The recommended degree of formalization is medium (cf 48).
There should be some formalization between the organizational units but less formalization within the units due to the high professionalization. Software Engineering has a medium capital requirement, which leads to medium formalization. Medium size organizations should have medium formalization. Medium formalization is consistent with the leadership style when top management's preference for microinvolvement is neither very great nor very low.
The recommended degree of centralization is medium (cf 45).
Software Engineering has an analyzer with innovation strategy. Centralization should be medium. There should be tight control over current activities and looser control over new ventures. Software Engineering is of medium size. Such organizations should have medium to high centralization. Medium centralization is recommended when top management has neither a great desire nor very little desire for microinvolvement.
Software Engineering's span of control should be narrow (cf 30).
It, too, at places should be moderate (cf 25).
Since Software Engineering has a nonroutine technology, it should have a narrow span of control.
Software Engineering should use media with high media richness (cf 85).
The information media that Software Engineering uses should provide a large amount of information (cf 85).
Incentives should be based on results (cf 85).
Software Engineering should use an undetermined process as means for coordination and control (cf 100).
When the environment of Software Engineering has high equivocality, high uncertainty, and high complexity, coordination and control should be obtained through integrators and group meetings. The richness of the media should be high with a large amount of information. Incentives must be results based. Coordination is a major issue requiring a lot of time by functional managers and product or project managers. Managers should make frequent adjustments in order to maintain project and product goals and use scarce functional resources and personnel efficiently. In an international firm, matrix dimensions will likely include country or region and may include either product, customer, or function. Project or product managers will likely be required to champion new innovations in customers, products or technologies. When the organization has a group climate, coordination should be obtained using integrators and group meetings. Incentives could be results based but with a group orientation. An organization with a group climate will likely have to process a large amount of information and will need information media with high richness.

ORGANIZATIONAL MISFITS

Organizational misfits compares the recommended organization with the current organization.
The following organizational misfits are present: (cf 100).
Current and prescribed configuration do not match.
Current and prescribed formalization do not match.

MORE DETAILED RECOMMENDATIONS

There are a number of more detailed recommendations (cf 100).

You may consider increasing the number of positions for which job descriptions are available.
You may consider supervising the employees more closely.
You may consider allowing employees less latitude from standards.
You may consider more written job descriptions.
Managerial employees may be asked to follow written instructions and procedures more closely.
You may consider having more written rules and procedures.

END

REPORT SUMMARY - Software Engineering

Time: 2:40:37 PM, 12/29/99
Scenario: Scenario 2

INPUT DATA SUMMARY

The description below summarizes and interprets your answers to the questions about your organization and its situation. It states your answers concerning the organization's current configuration, complexity, formalization, and centralization. Your responses to the various questions on the contingencies of age, size, technology, environment, management style, cultural climate and strategy factors are also given. The writeup below summarizes the input data for the analysis.
- Software Engineering has a matrix configuration (cf 100).
- Software Engineering has a small number of different jobs (cf 100).
- Of the employees at Software Engineering 76 to 100 % have an advanced degree or many years of special training (cf 100).
- Software Engineering has 3 to 5 vertical levels separating top management from the bottom level of the organization (cf 100).
- The mean number of vertical levels is 3 to 5 (cf 100).
- Software Engineering has 1 or 2 separate geographic locations (cf 100).
- Software Engineering's average distance of these separate units from the organization's headquarters is of no relevance because there is only one site undetermined (cf 100).
- An undetermined number of Software Engineering's total workforce is located at these separate units (cf 100).
- Job descriptions are available for operational employees, low and middle management (cf 100).
- Where written job descriptions exist, the employees are supervised closely to ensure compliance with standards set in the job description (cf 100).
- The employees are allowed to deviate a moderate amount from the standards (cf 100).
- 81 to 100 % non-managerial employees are given written operating instructions or procedures for their job (cf 100).
- The written instructions or procedures given are followed to a great extent (cf 100).
- Supervisors and middle managers are to a little extent free from rules, procedures, and policies when they make decisions (cf 100).
- More than 80 % of all the rules and procedures that exist within the organization are in writing (cf 100).
- Top Management is to some extent involved in gathering the information they will use in making decisions (cf 100).
- Top management participates in the interpretation of 41 to 60 % of the information input (cf 100).
- Top management directly controls 0 to 20 % of the decisions executed (cf 100).
- The typical middle manager has some discretion over establishing his or her budget (cf 100).
- The typical middle manager has some discretion over how his/her unit will be evaluated (cf 100).
- The typical middle manager has great discretion over the hiring and firing of personnel (cf 100).
- The typical middle manager has some discretion over personnel rewards - (ie, salary increases and promotions) (cf 100).
- The typical middle manager has some discretion over purchasing equipment and supplies (cf 100).
- The typical middle manager has some discretion over establishing a new project or program (cf 100).

- The typical middle manager has very great discretion over how work exceptions are to be handled (cf 100).
- Software Engineering has 25 employees (cf 100).
- Software Engineering's age is young (cf 100).
- Software Engineering's ownership status is public (cf 100).
- Software Engineering has few different products (cf 100).
- Software Engineering has few different markets (cf 100).
- Software Engineering only operates in one country (cf 100).
- Software Engineering has no different products in the foreign markets (cf 100).
- Software Engineering's major activity is categorized as service (cf 100).
- Software Engineering has a specialized customer-oriented service technology (cf 100).
- Software Engineering has a medium routine technology (cf 100).
- Software Engineering's technology is highly divisible (cf 100).
- Software Engineering's technology dominance is strong (cf 100).
- Software Engineering has either planned or already has an advanced information system (cf 100).
- Software Engineering's environment is complex (cf 100).
- The uncertainty of Software Engineering's environment is high (cf 100).
- The equivocality of the organization's environment is high (cf 100).
- Software Engineering's environment has an undetermined level of hostility (cf 100).
- Top management prefers to make policy and general resource allocation decisions (cf 100).
- Top management primarily prefers to make long-term decisions (cf 100).
- Top management has a preference for very aggregate information when making decisions (cf 100).
- Top management has a preference for some proactive actions and some reactive actions (cf 100).
- Top management is risk averse (cf 100).
- Top management has a preference for high control (cf 100).
- Software Engineering operates in an industry with a medium capital requirement (cf 100).
- Software Engineering has a high product innovation (cf 100).
- Software Engineering has a high process innovation (cf 100).
- Software Engineering has a high concern for quality (cf 100).
- Software Engineering's price level is undetermined relative to its competitors (cf 100).
- The level of trust is high (cf 100).
- The level of conflict is low (cf 100).
- The employee morale is high (cf 100).
- Rewards are given in a inequitably fashion (cf 100).
- The resistance to change is not known (cf 100).
- The leader credibility is high (cf 100).
- The level of scapegoating is low (cf 100).

THE SIZE

The size of the organization - large, medium, or small - is based upon the number of employees, adjusted for their level of education or technical skills.
Based on the answers you provided, it is most likely that your organization's size is medium (cf 50).
More than 75 % of the people employed by Software Engineering have a high level of education. Adjustments are made to this effect. The adjusted number of employees is lower than 500 but greater than 100 and Software Engineering is categorized as medium. However, for this adjusted number this size does not have a major effect on the organizational structure.

THE CLIMATE

The organizational climate effect is the summary measure of people and behavior.
Based on the answers you provided, it is most likely that the organizational climate is a group climate (cf 82).
It could also be the that climate is a developmental (cf 80).

The group climate is characterized as a friendly place to work where people share a lot of themselves. It is like an extended family. The leaders, or head of the organization, are considered to be mentors and, perhaps even parent figures. The organization is held together by loyalty or tradition. Commitment is high. The organization emphasizes the long-term benefit of human resource development with high cohesion and morale being important. Success is defined in terms of sensitivity to customers and concern for people. The organization places a premium on teamwork, participation, and consensus.
When the organization has a high level of trust it is likely that the organization has a group climate. An organization with little conflict can be categorized to have group climate. Employees with a high morale is one element of group climate. High leader credibility characterizes an organization with a group climate. An organization with a low level of scapegoating may have a group climate.
The developmental climate is characterized as a dynamic, entrepreneurial and creative place to work. People stick their necks out and take risks. The leaders are considered to be innovators and risk takers. The glue that holds organizations together is commitment to experimentation and innovation. The emphasis is on being on the leading edge. Readiness for change and meeting new challenges are important. The organization's long-term emphasis is on growth and acquiring new resources. Success means having unique and new products or services and being a product or service leader is important. The organization encourages individual initiative and freedom.
When the organization has a high to medium level of trust it is likely that the organization has a developmental climate. An organization with low level of conflict can be categorized to have a developmental climate. Employees with a high morale is frequently one element of a developmental climate. Medium to high leader credibility characterizes an organization with a developmental climate. An organization with a medium level of scapegoating may have a developmental climate.

THE MANAGEMENT STYLE

The level of management's microinvolvement in decision making is the summary measure of management style. Leaders have a low preference for microinvolvement; managers have a high preference for microinvolvement.
Based on the answers you provided, it is most likely that your management profile has a low preference for microinvolvement (cf 72).
It could also be that your management profile has an inappropriate preference (cf 70).
It could also be that your management profile has a high preference (cf 69).
The management of Software Engineering has a preference for delegating decisions. This will lead toward a low preference for microinvolvement. Management has a long-term horizon when making decisions, which characterizes a preference for a low microinvolvement. Since the management has a preference for making decisions on the basis of very aggregate information a low preference for microinvolvement characterization is appropriate.
The management dimensions are not in balance. This is likely to result in an ineffectual individual.
Management is risk averse. This is one of the characteristics of a manager with a high preference for microinvolvement. Management has a preference for using control to coordinate activities, which leads toward a high preference for microinvolvement.

THE STRATEGY

The organization's strategy is categorized as one of either prospector, analyzer with innovation, analyzer without innovation, defender, or reactor. These categories follow Miles and Snow's typology. Based on your answers, the organization has been assigned to a strategy category. This is a statement of the current strategy; it is not an analysis of what is the best or preferred strategy for the organization.
Based on the answers you provided, it is most likely that your organization's strategy is a prospector strategy (cf 73).
It could also be: a defender (cf 72).
It could also be: an analyzer with innovation (cf 72).
An organization with a prospector strategy is an organization that continually searches for market opportunities and regularly experiments with potential responses to emerging environmental trends. Thus, the organization is often the creator of change and uncertainty to which its competitors must respond.

However, because of its strong concern for product and market innovation, a prospector usually is not completely efficient.
With a concern for high quality a prospector strategy is a likely strategy for Software Engineering. With top management preferring a relatively low level of microinvolvement, the strategy is likely to be prospector.
An organization with a defender strategy is an organization that has a narrow product market domain. Top managers in this type of organization are expert in their organization's limited area of operation but do not tend to search outside their domains for new opportunities. As a result of this narrow focus, these organizations seldom need to make major adjustments in their technology, structure, or methods of operation. Instead, they devote primary attention to improving the efficiency of their existing operations.
Software Engineering has few products. It needs to defend these products well in the marketplace. Viability depends on being successful with these limited activities. With a concern for high quality a defender strategy is a likely strategy for Software Engineering.
An organization with an analyzer with innovation strategy is an organization that combines the strategy of the defender and the prospector. It moves into the production of a new product or enters a new market after viability has been shown. But in contrast to an analyzer without innovation, it has innovations that run concurrently with the regular production. It has a dual technology core.
An organization with a medium capital investment is likely to have some capabilities rather fixed, but can also adjust. The analyzer with innovation which seeks new opportunities but also maintains its profitable position is appropriate. For a medium routine technology, Software Engineering has some flexibility. It is consistent with an analyzer with innovation strategy. With a concern for high quality an analyzer with innovation strategy is a likely strategy for Software Engineering.

THE CURRENT ORGANIZATIONAL CHARACTERISTICS

Based on your answers, the organization's complexity, formalization, and centralization have been calculated. This is the current organization. Later in this report, there will be recommendations for the organization.
The current organizational complexity is medium (cf 100).
The current horizontal differentiation is medium (cf 100).
The current vertical differentiation is low (cf 100).
The current spatial differentiation is low (cf 100).
The current centralization is medium (cf 100).
The current formalization is high (cf 100).
The current organization has been categorized with respect to formalization, centralization, and complexity. The categorization is based on the input you gave and does not take missing information into account.

SITUATION MISFITS

A situation misfit is an unbalanced situation among the contingency factors of management style, size, environment, technology, climate, and strategy.
The following misfits are present: (cf 100).
Software Engineering has both a prospector strategy and a risk adverse management. This strategy conflicts with the management's risk adverse attitude. A prospector strategy demands a projection into the unknown with new and innovative products and services, where the returns are uncertain. A risk adverse management will be very uncomfortable with this high level of risk. Risk adverse managers prefer situations with less uncertainty. It is possible to either change the prospector strategy or hire more risk assuming managers. Usually a risk adverse management will control expenditures to reduce or eliminate the prospector projects. If the environment and markets call for a prospector strategy, a new management would be preferable. Some risk adverse managers can adapt, but it is very difficult.
Software Engineering has both a prospector strategy and not many products or markets. The prospector will create a broad range of new possible products and services, which requires a large number of possible products and markets. A prospector requires variety to explore and find new products and markets for its innovations. With limited product and market opportunity, the range of prospector possibilities may exceed the environmental possibilities. The prospector needs to seek new markets as well as new products. If the markets do not exist or cannot be created, the prospector will incur high costs of innovation without return.

Software Engineering has a group climate. This is a mismatch with a prospector strategy! A group climate has low resistance to change. A prospector strategy is committed to changes.

ORGANIZATIONAL CONSULTANT RECOMMENDATIONS

Based on your answers about the organization, its situation, and the conclusions with the greatest certainty factor from the analyses above Organizational Consultant has derived recommendations for the organization's configuration, complexity, formalization, and centralization. There are also recommendations for coordination and control, the appropriate media richness for communications, and incentives. More detailed recommendations for possible changes in the current organization are also provided.

ORGANIZATIONAL CONFIGURATIONS

The most likely configuration that best fits the situation has been estimated to be an adhocracy configuration (cf 68).
It is certainly not: a professional bureaucracy (cf -73).
It is certainly not: a machine bureaucracy (cf -73).
An adhocracy organization is normally an organization with high horizontal differentiation, low vertical differentiation, low formalization, decentralization, and great flexibility and responsiveness.
An adhocracy configuration is appropriate when neither the environmental equivocality of Software Engineering nor the environmental uncertainty is low. When the organization is also young, the conclusion that it should bean adhocracy is further strengthened. Since top management has a low preference for microinvolvement, the ad hoc configuration is feasible. However, the size of the organization is not very important for the choice of an adhocracy configuration. A prospector like Software Engineering should be configured as an ad hoc organization. An organization with a group climate could have an ad hoc configuration.
Since the organization has a prospector strategy, it cannot have a configuration like a professional bureaucracy.
When the organization has a prospector strategy, it cannot be a machine bureaucracy!

ORGANIZATIONAL CHARACTERISTICS

The recommended degree of organizational complexity is medium (cf 54).
Medium size organizations should have medium organizational complexity. Software Engineering has a technology that is somewhat routine, which implies that the organizational complexity should be medium. Because Software Engineering has an advanced information system, organizational complexity can be greater than it could otherwise. A group climate in the organization requires a medium level of complexity with a low level of vertical differentiation.
The recommended degree of horizontal differentiation is low (cf 34).
It, too, could be: medium (cf 24).
The recommended degree of vertical differentiation is low (cf 72).
It, too, could be: medium (cf 62).
The recommended degree of formalization is low (cf 56).
Software Engineering has a prospector strategy. A low formalization is required so that the organization can react quickly. Low formalization is also required because of the need for innovations. Since the set of variables in the environment that will be important is not known and since it is not possible to predict what will happen, no efficient rules and procedures can be developed, which implies that Software Engineering's formalization should be low. Low formalization is consistent with top management having a low preference for microinvolvement. A group climate in the organization requires a low level of formalization.
The recommended degree of centralization is low (cf 46).
There is evidence against it should be: high (cf -16).
Software Engineering has a prospector strategy. A low centralization is required so that the organization can react and innovate quickly. Since there are many factors in the environment that affect the organization but Software Engineering does not know which factors are or will be important for Software Engineering,

centralization should be low. Low centralization can be allowed when top management has no desire for microinvolvement. A group climate in the organization requires a low level of centralization.
Software Engineering's span of control should be moderate (cf 62).
Since Software Engineering has some technology routineness, it should have a moderate span of control.
Software Engineering should use media with high media richness (cf 85).
The information media that Software Engineering uses should provide a large amount of information (cf 85).
Incentives should be based on results (cf 85).
Software Engineering should use meetings as means for coordination and control (cf 94).
When the environment of Software Engineering has high equivocality, high uncertainty, and high complexity, coordination and control should be obtained through integrators and group meetings. The richness of the media should be high with a large amount of information. Incentives must be results based. An open organizational climate and team spirit must be fostered. Information must be shared among all levels. Constructive conflict on 'what to do' will be usual. Individual tolerance of ambiguity and uncertainty will be necessary. Individual performance evaluation will be problematic and largely subjective. Mutual adjustments of 'give and take' will be the norm. Frequent informal meetings and temporary task forces will be the primary coordinating devices. When the organization has a group climate, coordination should be obtained using integrators and group meetings. Incentives could be results based but with a group orientation. An organization with a group climate will likely have to process a large amount of information and will need information media with high richness.

ORGANIZATIONAL MISFITS

Organizational misfits compare the recommended organization with the current organization.
The following organizational misfits are present: (cf 100).
Current and prescribed configuration do not match.
Current and prescribed centralization do not match.
Current and prescribed formalization do not match.

MORE DETAILED RECOMMENDATIONS

There are a number of more detailed recommendations (cf 100).
You may consider supervising the employees less closely.
You may consider fewer written job descriptions.
Managerial employees may be asked to pay less attention to written instructions and procedures.
You may give supervisors and middle managers fewer rules and procedures.
You may consider having fewer rules and procedures put in writing.

END

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

SIMULATION REPORTS

The following chart (Fig C.1) presents the simulated organization and the simulated software process. The process presents only four cycles of evolution. Each cycle has the activities described in Chapter VII (Fig. 7.1).

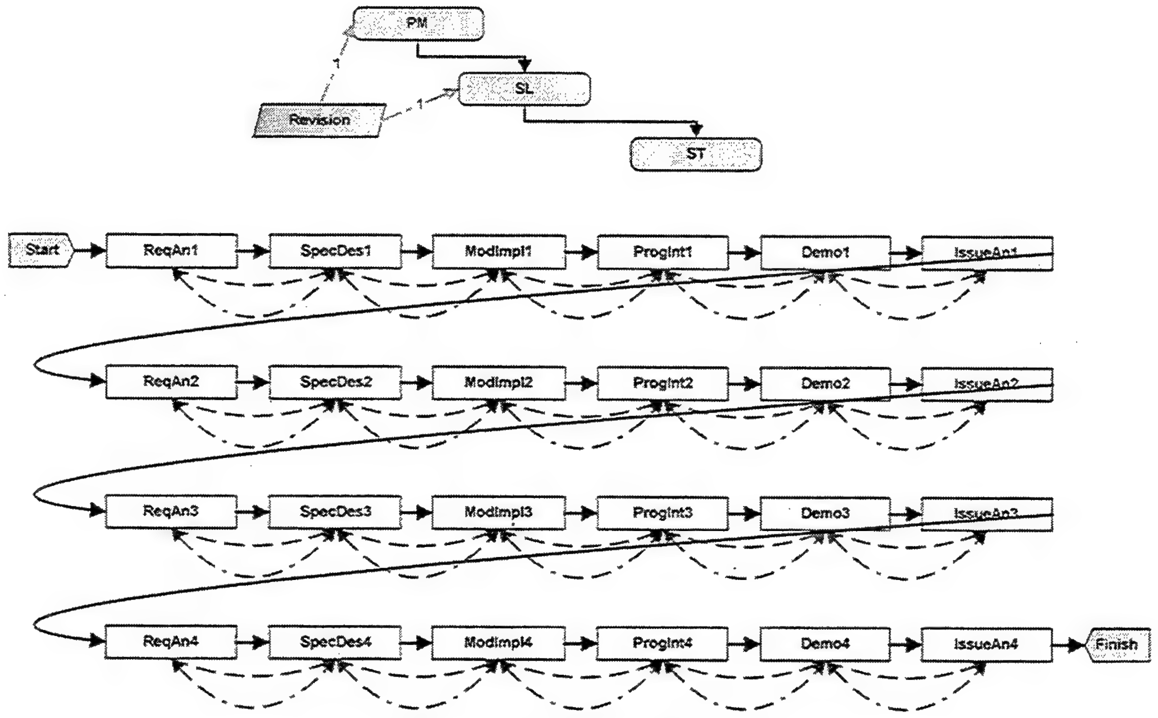


Figure C.1: Project layout[1].

[1] Note: The detailed description of the notation can be found on the VitéProject user manual (Levitt, 1999). Rectangles indicate tasks. Rounded-corner rectangles indicate roles. Parallelograms indicate meetings. Double-headed-dashed arrows indicate information dependencies between tasks. Dashed arrows indicate problem dependencies between tasks. Normal arrows indicate precedence dependencies between tasks.

1. Simulation Results

Table C.1 shows the expected durations and the standard deviations in days for the sixteen scenarios simulated. The simulations were configured to run 30 cases for each scenario. The column titles LGC shows the complexity measure for each scenario measured in LGC.

Table C.1: Simulation results

Scenario	*Efficiency*	*Req. Vol.*	*Complexity*	*E(t) days*	*SD(t) days*	*LGC*
LLL	L	L	L	88	5	746
LLH	L	L	H	101	6	781
LLH2.5	L	L	H2.5	254	16	1334
LLH5	L	L	H5	507	31	3230
LHL	L	H	L	101	7	746
LHH	L	H	H	128	10	781
LH2.5	L	H	H2.5	319	25	1334
LH5	L	H	H5	638	49	3230
HLL	H	L	L	32	2	746
HLH	H	L	H	42	3	781
HLH2.5	H	L	H2.5	105	7	1334
HLH5	H	L	H5	209	14	3230
HHL	H	H	L	42	3	746
HHH	H	H	H	49	4	781
HHH2.5	H	H	H2.5	122	9	1334
HHH5	H	H	H5	244	18	3230

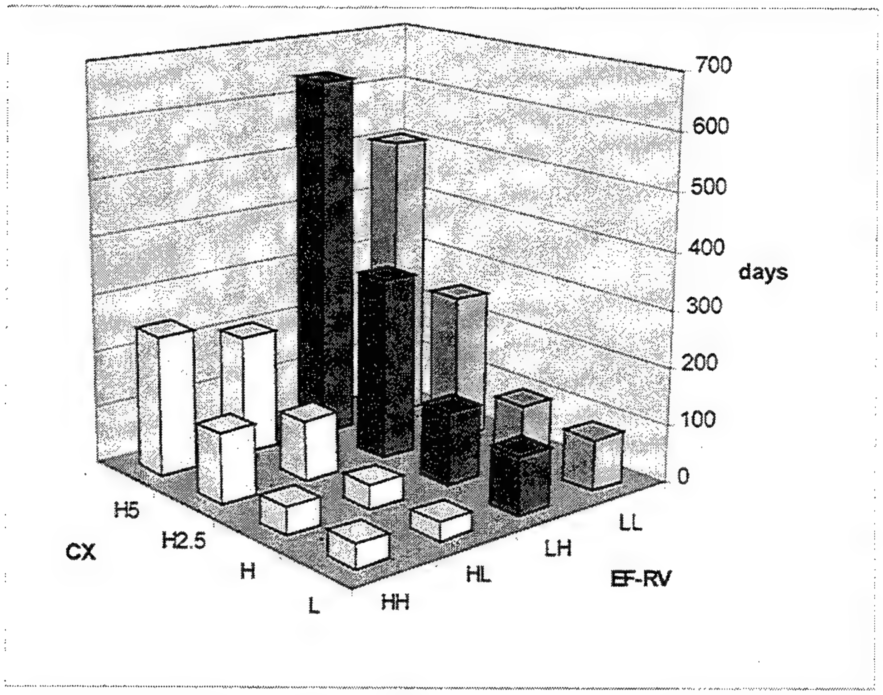


Figure C.2: Effects of complexity

Figure C.2 shows the effects of the complexity in the expected duration of similar efficiency and requirements volatility scenarios. Observe that the effect of complexity is different when the efficiency and requirement volatility vary.

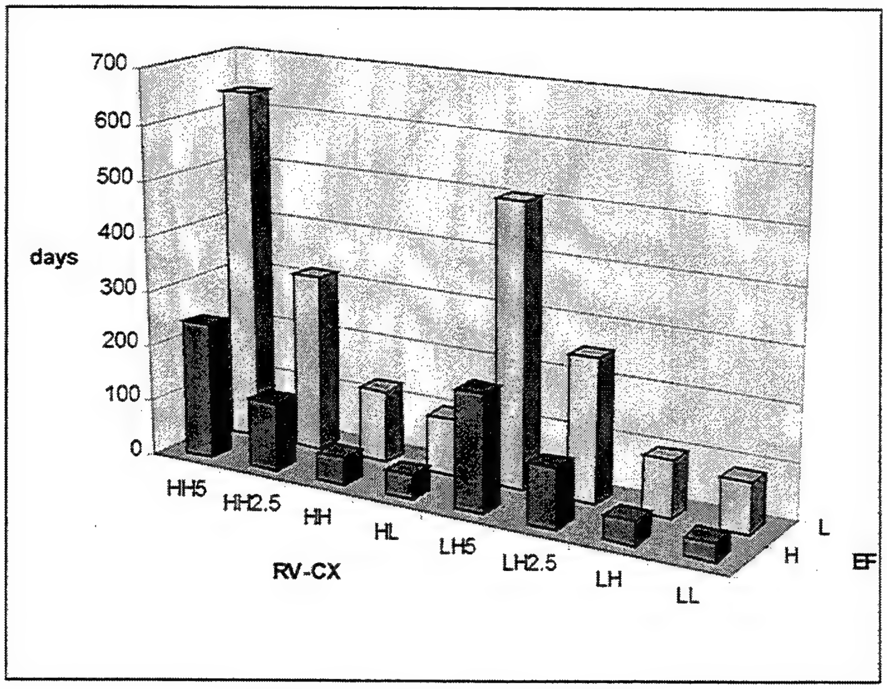


Figure C.3: Effects of efficiency

Figure C.3 shows the effects of efficiency. For same values of complexity and requirements volatility, the durations for high efficiency scenarios were 40% of the durations for low efficiency ones.

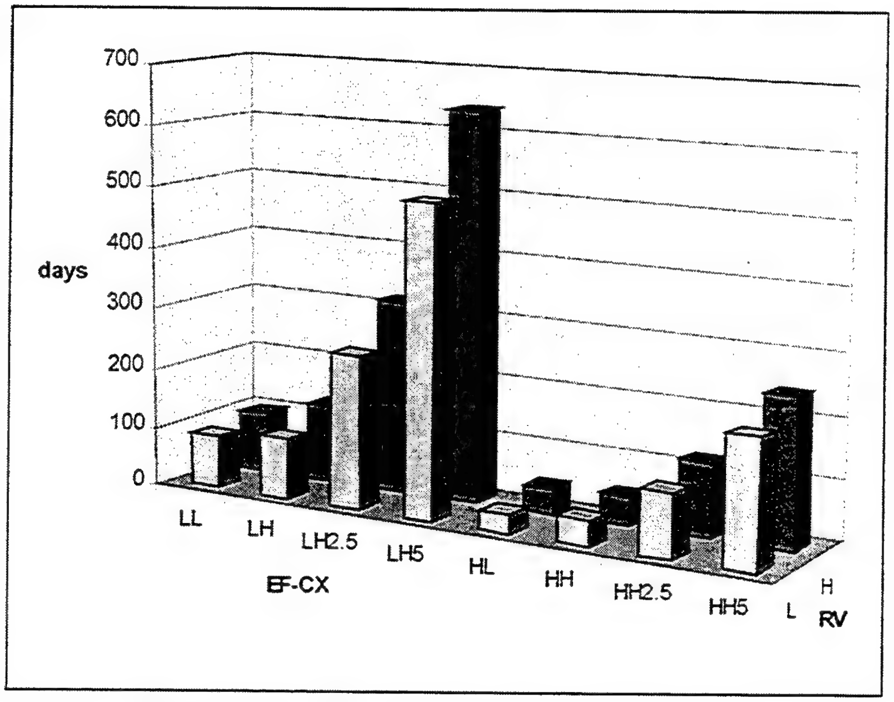


Figure C.4: Effects of requirement volatility

Figure C.4 shows the effects of requirements volatility. For same values of complexity and efficiency, the durations for volatile scenarios were 122% of the durations for non volatile ones.

APPENDIX D

PARAMETER CONFIGURATION FOR VITEPROJECT

VitéProject uses a set of default values for the variables of the model. These values are stored in a file named "behmatrx.opd" in the subdirectory of VitéProject. The behavior of the model depends on the values of these variables that are collectively called Behavior Matrix. This Appendix discusses the concepts considered in the behavior matrix and their relationship with software projects.

(1) Participant attention rule: Defines the probability distribution applied to the different selection methods (e.g. priority, FIFO, LIFO, random) of picking items to process.

(2) Participant tool selection rules: Defines the probability distribution applied to different information exchange tools (e.g. conversation, email, fax, memo, phone, video, voice-mail) given the type of message (e.g. Exception, Decision, etc.) A tool selected for an information exchange determines (1) the time needed for the message to move from one participant to another and (2) the time the message will stay in the in-tray of the receiver participant.
Findings:
   i. Even if there is one matrix for each role, all the matrices are identical.
   ii. Too much emphasis on voicemail. We expected more weight on conversation, phone and email.

(3) Activity Verification Failure Probability (VFP) adjustment: There are two VFP (internal and external). The internal VFP depends on the complexity of the requirement and the skills of the participants. The external VFP depends on the

complexity of the solution and the skills of the participants. The processing speed of responsible participants is affected by the solution complexity and the requirement complexity.

(4) Activity Information Exchange Frequency adjustment: This adjustment depends on the uncertainty of the activity and the team experience.

(5) Participant Processing Speed adjustment: This adjustment depends on the match between the participant and activity skill requirements.

(6) Definition of Rework, Quick-Fix, and Ignore decisions: This matrix defines how much of the original failed work should be reworked, quick-fixed or ignored. The values depend on the following failure types:

   i. Internal|Internal: Amount of rework of an activity given internal activity failure (based on VFPInternal.).
   ii. Internal|External: Amount of rework of an activity given external failure (based on VFP External.).
   iii. Internal|External: Amount of rework of a failure dependent activity given external failure of an independent activity (based on VFP External of the independent activity.).

(7) Impact of participant information exchange behavior on its VFP: This adjustment depends on the attendance or non-attendance of the participant to information exchange events related to the activities.

(8) Impact of participant decision-making behavior on the VFP of failed activity: This adjustment depends on the centralization level of the organization.

(9) Finally there is a set of matrices to implement Project Decision Making Policies including how to determine to whom to report an exception, how to make a decision for an exception, what is the maximum time a participant will wait before it takes delegation by default.

The following source code is the behavior matrix provided as default by ViteProject.

```
%==========================================================================%
%
%
% BehMatrx.opd - Vité-Project uses default qualitative-to-quantitative calibration
% parameter values defined in this file.  To override any of the default calibration
% values, place a modified version of this file in the directory that holds Vité-Project
% and specify the file name in the Vité-Project simulation control dialog box.  Vité-
% Project will load this file automatically.
%
% Each matrix defines an association set: the row selection, when associated with the
% column selection, has the behavior of the corresponding matrix value.  For example, for
% the ParticipantAttentionRule, a Project Manager (PM) will select an item from the
% intray by Priority with probability 0.5.  Notation:
% PM = Project Manager
% SL = participant subteam leader
% ST = participant
%
% Revisions:
%  10.17.97 Update comments and values
%==========================================================================%

(Application BehMatrices)

%==========================================================================
% Participant attention rule: - A participant uses this attention rule to select an item
% from its in-tray.  By default, all participants in Vité-Project share this common
% attention rule.
% Example: a Project Manager (PM) will select an item from the intray by priority with
% probability 0.5, with FIFO with probability 0.1, etc.
%==========================================================================

(Matrix ParticipantAttentionRule
    :Row       PM SL ST                           %= Participant role.
    :Column    Priority FIFO LIFO Random      %= Item Selection strategy.
    :Values    (0.4 0.3 0.2 0.1)              %= Probability corresponding strategy
            (0.3 0.4  0.2  0.1)               % will be applied.
               (0.1 0.5  0.3  0.1)
)

%==========================================================================
%  Participant tool selection rules
%  Information exchange tool selection is based on only Message types (e.g,. Exception,
%  Decision, etc.)  A tool selected for an information exchange determines (1) the time
%  needed for the message to move from one participant to another and (2) the time the
%  message will stay in the in-tray of the receiver participant.
%  Example: Given an exception to process, the PM will never choose the Phone or Video.
%  Note that Decisions go directly to the recipient in-tray without use of a information
%  exchange tool.
%==========================================================================

%-------------------------------------------------------------------
%  This rule only applies to project managers
%
(Matrix ToolSelectionRulesPM
    :Row [Message type]: Decision Exception InfoExchange    Meeting Noise
    :Column    [Tool to use]: Conversation Email Fax Memo Phone Video VoiceMail
    :Values    (0.15 0.20 0.20 0.20 0.0  0.0 0.25)   %= Probability
               (0.20 0.20 0.20 0.20 0.1  0.0 0.10)   % a specific tool
               (0.25 0.1  0.1  0.15 0.25 0.0 0.15)   % will be used
               (0.5  0.0  0.0  0.0  0.2  0.0 0.3)
               (0.3  0.1  0.05 0.1  0.35 0.0 0.1)
)
%-------------------------------------------------------------------
```

```
%  This rule only applies to participant leaders
%
(Matrix ToolSelectionRulesSL
    :Row        Decision Exception InfoExchange Meeting Noise
    :Column     Conversation Email Fax Memo Phone Video VoiceMail
    :Values     (0.15 0.20 0.20 0.20 0.0  0.0 0.25)   %= Probability
                (0.20 0.20 0.20 0.20 0.1  0.0 0.10)   % a specific tool
                (0.25 0.1  0.1  0.15 0.25 0.0 0.15)   % will be used
                (0.5  0.0  0.0  0.0  0.2  0.0 0.3)
                (0.3  0.1  0.05 0.1  0.35 0.0 0.1)
)
%------------------------------------------------------------------
%  This rule only applies to sub teams
%
(Matrix ToolSelectionRulesST
    :Row        Decision Exception InfoExchange Meeting Noise
    :Column     Conversation Email Fax Memo Phone Video VoiceMail
    :Values     (0.15 0.20 0.20 0.20 0.0  0.0 0.25)   %= Probability
                (0.20 0.20 0.20 0.20 0.1  0.0 0.10)   % a specific tool
                (0.25 0.1  0.10 0.15 0.25 0.0 0.15)   % will be used
                (0.5  0.0  0.0  0.0  0.2  0.0 0.3)
                (0.3  0.1  0.05 0.10 0.35 0.0 0.10)
)

%=========================================================================
%  Activity Verification Failure Probability (VFP) adjustment:
%     The formula used to determine activities' internal and external VFP:
%
% ?activity.VFPexternal =
%    ?proj.VFPexternal * SolutionComplexityEffect * ParticipantSkillEffect;
% ?activity.VFPinternal =
%     ?proj.VFPinternal * RequirementComplexityEffect * ParticipantSkillEffect;
%
%     The adjustment coefficients (e.g., SolutionComplexityEffect ParticipantSkillEffect)
% are determined by values in the following matrices.

%=========================================================================

%------------------------------------------------------------------
%  Effect of Activity solution complexity on processing speed of responsible
%  participants.
%
(Matrix SolutionComplexityEffect
    :Row        High Medium Low       %= Level of solution complexity.
    :Values     1.5 1.0 0.67    %= Value of SolutionComplexityEffect
)

%------------------------------------------------------------------
%  Effect of Activity requirement complexity on responsible participant processing speed.
%
(Matrix RequirementComplexityEffect
    :Row        High Medium Low       %= Level of requirement complexity.
    :Values     1.5 1.0 0.67          %= Value of RequirementComplexityEffect
)

%------------------------------------------------------------------
%  Effect of Participant-Activity skill match on activity VFP:
%  If responsible participant skill matches the skill requirement of the
%  corresponding activity, then use this matrix to determine
%  ParticipantSkillEffect.
%
(Matrix ParticipantSkillMatchVFP
    :Row        High Medium Low       %= Level of participant App. Experience
    :Column     High Medium Low       %= Participant Required Skill Level.
    :Values     (0.5 0.7 0.9)         %= Values of ParticipantSkillEffect.
                (0.7 1.0 1.2)
                (0.9 1.2 1.5)
)

%------------------------------------------------------------------
%  Effect of Participant-Activity match on activity VFP:
```

```
%  If participant skill DOES NOT match activity's skill requirement, then
%  use this matrix to determine ParticipantSkillEffect.  Failure of
%  participant-activity skill match creates a major VFP penalty.
%
(Matrix ParticipantSkillNonMatchVFP
    :Row        High Medium Low       %= Level of participant App. Experience
    :Column     High Medium Low       %= Participant other Skill Level.
    :Values     (2.0 2.0 2.0)         %= Values of ParticipantSkillEffect.
                (2.5 2.5 2.5)
                (3.5 3.5 3.5)
)

%============================================================================
%  Activity Information Exchange Frequency adjustment: The following formula is used to
%  determine probabilistic information exchange frequency of an activity
%
% ?activity.InfoExchangeFrequency = ?proj.InfoExchangeFrequency *
%  ActivityUncertaintyEffect * TeamExperienceEffect
%============================================================================

%----------------------------------------------------------------------
%  Effect of Activity uncertainty on information exchange frequency:
%
(Matrix ActivityUncertaintyEffect
    :Row        High Medium Low       %= Level of activity uncertainty
    :Values     1.4 1.00 0.67  %= Value of ActivityUncertaintyEffect
)

%----------------------------------------------------------------------
%  Effect of responsible Participant team experience on information exchange frequency:
%
(Matrix TeamExperienceEffect
    :Row        High Medium Low       %= Level of participant team experience.
    :Values     0.67 1.0 1.5   %= Value of TeamExperienceEffect
)

%============================================================================
%  participant processing speed adjustment:
%     The following formula determines participant processing speed.  Since participant
%     processing speed is based on its match with the skill requirement of its assigned
%     activity, the ParticipantSpeed is associated with each activity.  (Vité-Project
%     assumes that each activity can have only ONE responsible participant working on
%     it.)
%
% ?activity.ResponsibleParticipantSpeed =
%          1.0 / (?Participant.NumberOfParticipants * ?Participant.APS0 *
% ParticipantSkillEffect * ?Participant.TimePercentageForProject);
%
%     The rule uses 1/ "time needed to process a work unit" to calculate speed.
%============================================================================

%----------------------------------------------------------------------
%  Effect of Participant-Activity match on APS:
%  If responsible participant skill matches the activity's skill
%  requirement, then use this matrix to determine the value of
%  ParticipantSkillEffect.
%
 (Matrix ParticipantSkillMatchAPS
    :Row        High Medium Low       %= Level of participant App.Experience.
    :Column     High Medium Low       %= Participant Required Skill level.
    :Values     (2.0 1.5 0.9)         %= Values of ParticipantSkillEffect
                (1.5 1.0 0.7)
                (0.9 0.7 0.5)
)

%----------------------------------------------------------------------
%  If participant skill DOES NOT match activity's skill requirement, then
%  use this matrix to determine the value of ParticipantSkillEffect.
%
(Matrix ParticipantSkillNonMatchAPS
    :Row        High Medium Low       %= Level of participant App. Experience.
    :Column    High Medium Low        %= Participant Other Skill level.
    :Values    (0.7 0.7 0.7)          %= Values of ParticipantSkillEffect
               (0.5 0.5 0.5)
               (0.3 0.3 0.3)
)

%============================================================================
%  Definition of Rework, Quick-Fix, and Ignore decisions:
%  This matrix defines how much of the original failed work should be reworked based
%  decision types (i.e., Reworked, Quick-Fixed, Ignore).  The actual rework volume is the
%  given subactivity volume * % - of failed work that needs to be reworked * user-
%  interface defined "Strength" of failure dependent activity relationship
%
%    The values change depending on the failure types described below:
%
%    Internal!Internal: Amount of rework of an activity given internal activity failure
%    (based on VFPInternal.)
%
%    Internal!External: Amount of rework of an activity given external failure (based on
%    VFP External.)
%    Internal!External Amount of rework of a failure dependent activity given external
%    failure of an independent activity (based on VFP External of the independent
%    activity.)
%
%============================================================================
%
(Matrix ReworkVolume
    :Row       Internal Internal!External External!External %= failure type
    :Column    Rework Quick-Fix Ignore       %= Decision for the exception
    :Values    (1.0 0.5 0.0)                 %= Percent of failed work
               (1.0 0.5 0.0)                 % that needs to be reworked.
               (1.0 0.5 0.0)
)

%============================================================================
%  Impact of participant information exchange behavior on its VFP:
%  Vité-Project simulates the impact of participant information exchange behavior on its
%  VFP by updating VFP based on the effect weight as shown below (same for VFPexternal
%  and VFPinternal):
%
%  ?activity.VFPinternal = ?activity.VFPinternal * VFPInfoXEffect;
%  if   ?activity.VFPinternal > 1.0;
%  then ?activity.VFPinternal = 1.0;
%
%  The value of VFPInfoXEffect is retrieved from the following matrices.
%
%  VFP updating is dynamic, i.e., it happens whenever an information exchange finishes.
%  You can disable the effects by setting matrix values to 1.0.
%============================================================================

%----------------------------------------------------------------------
%  This matrix defines the weight for updating participant verification failure
probabilities (*** Internal and External) due to not attending to information exchange
with peers, meetings and noise respectively.
%  NOTE: Weight =1.0 implies no effect of ignoring communications, meetings or noise.
%
(Matrix ParticipantNonAttendanceFailureEffect
    :Row       InfoXNonAttend MeetNonAttend NoiseNonAttend
                                     %= Nonatt InfoX type
    :Column    High Medium Low              %= Level of formalization
    :Values    (1.01 1.07 1.1)              %= VFPInfoXEffect.
            . (1.10 1.07 1.05)
               (1.0  1.00 1.00)
)

%----------------------------------------------------------------------
%  This matrix defines the weight for updating participant verification failure
%  probability due to attending to information exchange from peers, meetings
%  and noise respectively.
%
```

```
(Matrix ParticipantAttendanceFailureEffect
    :Row       InfoXAttend MeetAttend NoiseAttend %= Nonatt InfoX type
    :Column    High Medium Low                        %= Level of formalization
    :Values    (0.99  0.96  0.95)                 %= VFPInfoXEffect.
         (0.90  0.96  0.99)
               (1.0   1.0   1.0)
)

%==========================================================================
%  Impact of participant decision-making behavior on the VFP of failed activity:
%  Vité-Project simulates the impact of participant information exchange behavior on its
%  VFP  updating VFP based on the effect weight as shown below
%  (same for VFPexternal):
%
%        ?activity.VFPinternal = ?activity.VFPinternal * VFPInfoXEffect;
%        if   ?activity.VFPinternal > 1.0;
%        then ?activity.VFPinternal = 1.0;
%
%     The value of VFPInfoXEffect is retrieved from the following matrices, based
%     decision-maker's role and the type of decision it has made.
%
%     VFP updating is dynamic, i.e., it happens whenever a decision is made.
%
%     You can turn off the effects by setting values of the matrices to 1.0.
%==========================================================================

%-----------------------------------------------------------------------
%  Matrix used for Low centralization:
%
(Matrix LowCentralDecisionWeight
    :Row       PM SL ST                      %= Decision-maker's role.
    :Column    Rework Quick-Fix Ignore %= Type of decision made.
    :Values    (0.95 1.0 1.05)               %= VFPInfoXEffect for update VFP
               (0.95 1.0 1.05)
               (0.95 1.0 1.05)
)

%-----------------------------------------------------------------------
%  Matrix used for Medium centralization:
%
(Matrix MediumCentralDecisionWeight
    :Row       PM SL ST                      %= Decision-maker's role.
    :Column    Rework Quick-Fix Ignore %= Type of decision made.
    :Values    (0.9  0.95 1.05)              %= VFPInfoXEffect for update VFP
               (0.95 1.0 1.05)
               (0.95 1.05 1.1)
)

%-----------------------------------------------------------------------
%  Matrix used for High centralization:
%
(Matrix HighCentralDecisionWeight
    :Row       PM SL ST                      %= Decision-maker's role.
    :Column    Rework Quick-Fix Ignore %= Type of decision made.
    :Values    (0.9  0.95 1.05)              %= VFPInfoXEffect for update VFP
               (0.95  1.0 1.1)
               (0.95 1.1 1.2)
)

%==========================================================================
%  Following matrices are used to implement Project Decision Making Policies
%     including how to determine to whom to report an exception, how to make
%     a decision for an exception, what is the maximum time a participant will
%     wait before it takes delegation by default.
%
%==========================================================================

%-----------------------------------------------------------------------
%  Time To Wait For Decision Policy:
%     This matrix defines how long a participant should wait for a decision
%     before it assumes delegation by default. Participants playing different
%     roles in the organization may have different time-out durations.
%
(Matrix TimeToWaitForDecision
    :Row       PM SL ST                        %= Participant roles
    :Values     480                    %= Time-out duration in minutes
              960
              960
)

%--------------------------------------------------------------------
%  Decision Maker Policy:
%     This matrix is used by a participant to determine who should make
%     decision for his/her exception, based on project's centralization
%     policy. The assumption is that more centralized project teams
%     requires higher level participants make decisions for exceptions.
%
(Matrix DecisionMakerPolicy
    :Row       PM SL ST                %= Decision maker's role
    :Column    High Medium Low                 %= Level of centralization
    :Values    (0.6 0.2 0.1)           %= Probability
            (0.3 0.6 0.3)                % a certain role should
               (0.1 0.2 0.6)             % make the decision.
)

%--------------------------------------------------------------------
%  Decision Choice Policy:
%     This matrix is used by a decision-maker to determine how an exception should be
dealt with, based on project's centralization policy. NOTE:  The assumption is that
higher level participants (e.g., project managers) tend to make more Rework decisions.
Vité experience has found this assumption reasonable for routine engineering design.
However, for domains like software engineering, Vité staff has found that the reverse is
true.  Participants (hackers) want to fix every known bug, whereas managers want to ship
on time, even with known, non-serious bugs.  This matrix should be adjusted to reflect
the "bug fixing" culture of the organization being modeled.
%
(Matrix DecisionChoicePolicy
    :Row       PM SL ST                        %= Decision-maker's role
    :Column    Rework Quick-Fix Ignore %= Decision type
    :Values    (0.65  0.3 0.05)                %= Probability
            (0.4 0.4 0.2)              % the decision-maker will
               (0.05 0.35 0.6)                 % make a certain type of decision
)

%===========================================================================
%  Information exchange Probability adjustment:
%  The following matrices adjust the frequency probability of different types of
information exchange based on the Level of project Formalization:
%
% ?AdjustedInfoXProbability = OriginalCommunicationProbability * AdjustFactor;
%
% The Info Exchange AdjustFactor is retrieved from the following matrix given the level
of formalization.
%
% NOTE: Meeting frequency is not adjustable in Vité-Project, so the Meet row of the
matrices is not meaningful.
%
%===========================================================================

%--------------------------------------------------------------------
%  This matrix defines the VFP adjustment factor for different types of information
exchange.
%
(Matrix CoordinationDistribution
    :Row       InfoX Meet Noise                %= Information exchange type
    :Column    High Medium Low                 %= Level of formalization
    :Values    (0.5 1.0 2.0)                   %= Info Exchange AdjustFactor.
               (0.7 1.0 1.0)
               (1.0 1.0 1.0)
)

%===========================================================================
% In Vité-Project, when a participant picks up an information exchange item, it has to
% decide whether to attend the request for information exchange.  This matrix defines the
% chance a participant attends to a given type of information exchange given a level of
% strength of organization matrix.
%
% e.g., if Matrix Strength is High (as in a Project organization), then a participant
% will probabilistically attend to 80% of information exchanges, and 20% of the meetings
% and 20% of the Noise. Project organizations have high Matrix strength; functional teams
% have low matrix strength.
%
%
(Matrix CoordinationPriority
    :Row        InfoX Meet Noise                %= Type of information exchange
    :Column     High Medium Low                 %= Org Matrix Strength
    :Values     (0.9 0.7 0.6)                   %= Probability
                (0.6 0.7 0.9)                   % a participant will attend
                (0.2 0.2 0.2)                   % a communication.
)

%=============================================================================
%
% Communications-related matrices
%
%=============================================================================

%----------------------------------------------------------------------
%  This matrix defines the length of time (in minutes) it takes to
%  deliver messages using different communication tools
%

(Matrix ToolTimeToDeliver
    :Row        Conversation Email Fax Memo Phone VideoConf VoiceMail %= Communication
tool
    :Values     10
                1
                1
                5
                1
                1
                1
)

%----------------------------------------------------------------------
%  This matrix defines the length of time (in minutes) it takes for
%  messages to expire in the recipients in-tray
%

(Matrix ToolTimeToExpire
    :Row        Conversation Email Fax Memo Phone VideoConf VoiceMail %= Communication
tool
    :Values     60
                2400
                1440
                2400
                5
                10
                960
)

%----------------------------------------------------------------------
%  This matrix defines the volume (in minutes) for each type of message
%

(Matrix MessageVolume
    :Row        PM SL ST                        %= Recipients role
    :Column     decision exception info_exchange meeting noise %= Message type
    :Values     (10 120 30 0 10)
                (10 240 30 0 10)
                (10 240 30 0 10)
)
```

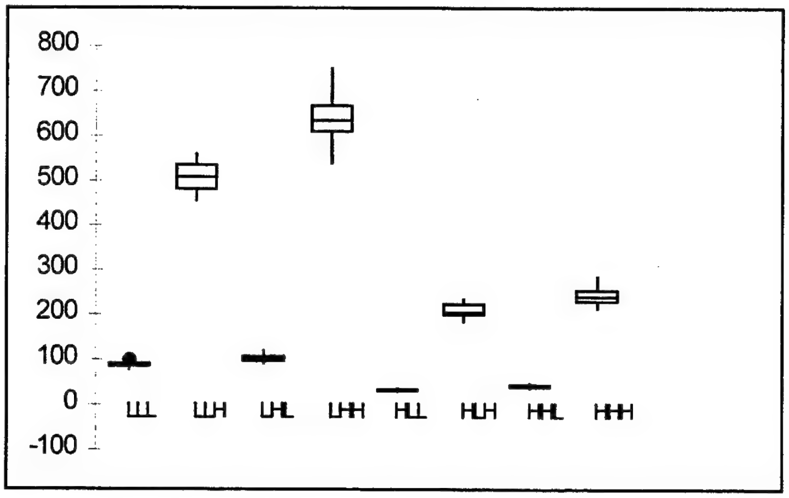
```
%%%%%%%%%%%%%%%
% END OF FILE %
%%%%%%%%%%%%%%%
```

APPENDIX E

STATISTICAL ANALYSIS OF SIMULATION OUTPUTS

A. Descriptive statistics and box plots

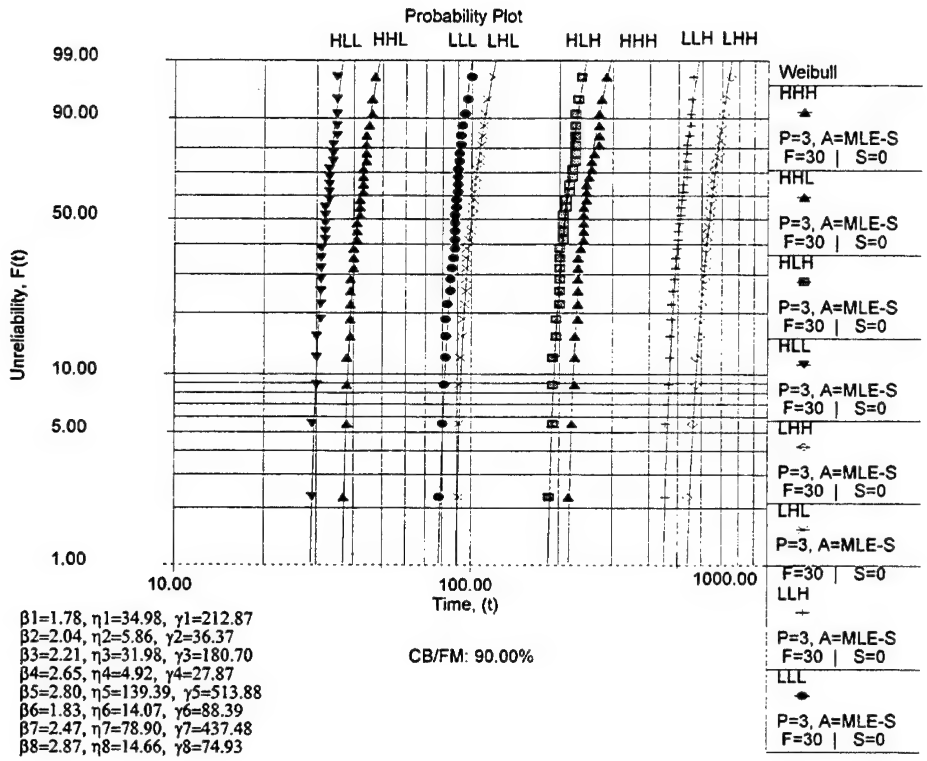
	LLL	*LLH*	*LHL*	*LHH*	*HLL*	*HLH*	*HHL*	*HHH*
Mean	88	507.3333	100.8667	638	32.23333	209	41.56667	244
Standard Error	0.91977	5.732211	1.323296	8.939773	0.334538	2.530276	0.495226	3.342516
Median	88	507.5	100.5	635	32	205	41.5	240
Mode	88	535	96	575	31	200	39	230
Standard Deviation	5.037788	31.39661	7.247988	48.96515	1.83234	13.85889	2.712466	18.30771
Kurtosis	0.103871	-1.16404	-0.23193	-0.17195	-0.95402	-1.13356	-0.91119	-0.56513
Skewness	0.130034	-0.00625	0.562342	0.203825	0.062831	0.138658	0.216263	0.600225
Range	22	105	28	210	6	50	10	70
Minimum	78	455	91	540	29	185	37	215
Maximum	100	560	119	750	35	235	47	285
Count	30	30	30	30	30	30	30	30
CI (95.0%)	1.881142	11.72369	2.706445	18.2839	0.684208	5.174999	1.012852	6.836217



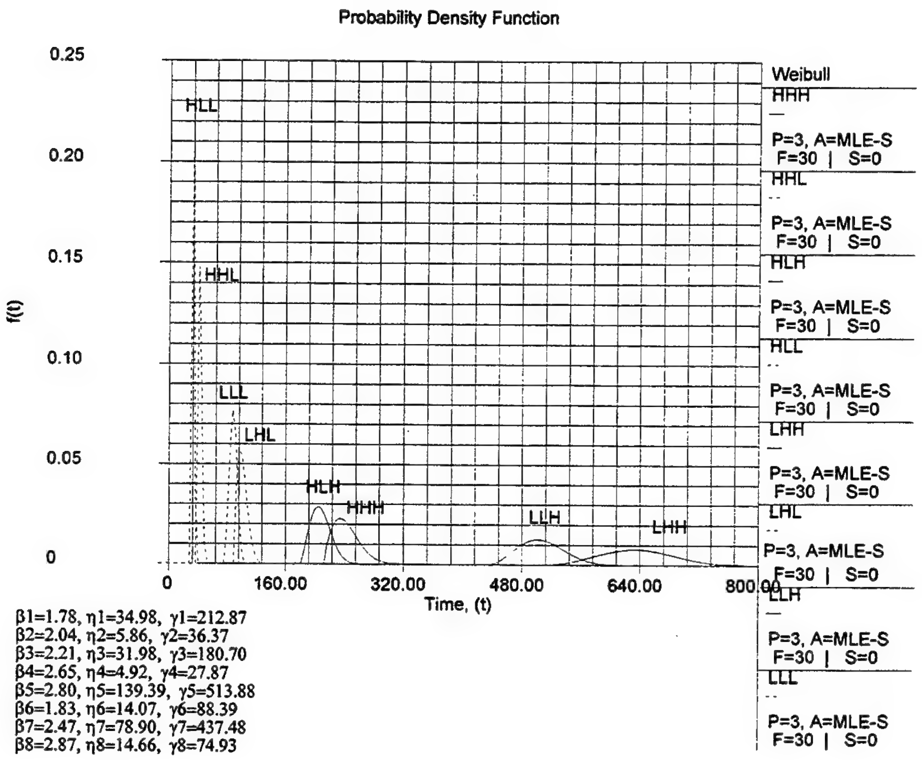
The descriptive statistics do not give conclusive information about the kind of distribution observed. The boxplots show that complexity (the third variable) has the strongest influence over the development time, efficiency seems to have less impact, and requirements volatility seems to have moderate influence.

B. Weibull probability plots

The data obtained from the simulations was analyzed with a statistical software package from Reliasoft. The product checks what is the distribution function that better fits the sample. The distributions compared were exponential (one and two parameters), Weibull (two and three parameters), normal, and lognormal. In all the cases the tool found that Weibull with three parameters was the best fit. The following plot is a Weibull paper and shows the data points as icons and the distribution function recommended by the tool as lines.



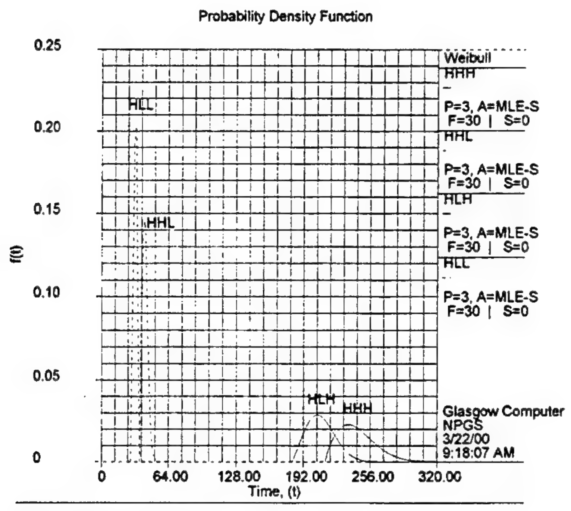
C. Probability distribution functions



$\beta 1=1.78, \eta 1=34.98, \gamma 1=212.87$
$\beta 2=2.04, \eta 2=5.86, \gamma 2=36.37$
$\beta 3=2.21, \eta 3=31.98, \gamma 3=180.70$
$\beta 4=2.65, \eta 4=4.92, \gamma 4=27.87$
$\beta 5=2.80, \eta 5=139.39, \gamma 5=513.88$
$\beta 6=1.83, \eta 6=14.07, \gamma 6=88.39$
$\beta 7=2.47, \eta 7=78.90, \gamma 7=437.48$
$\beta 8=2.87, \eta 8=14.66, \gamma 8=74.93$

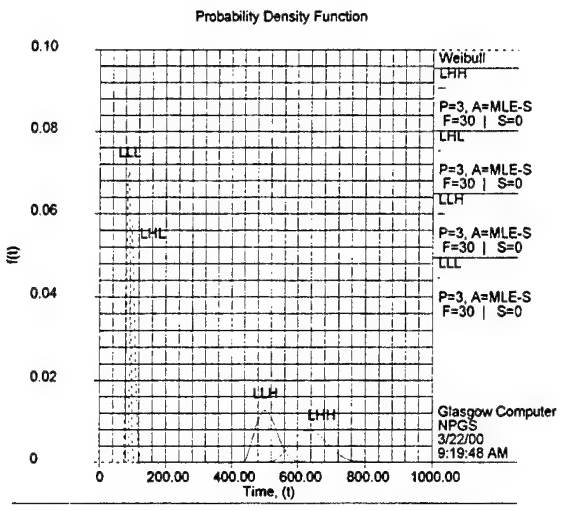
D. Effect of requirements volatility

The following graphs show the influence of requirements volatility. Two graphs are presented to discriminate the cases of high and low efficiency in order to avoid confounding factors.



$\beta1=1.78$, $\eta1=34.98$, $\gamma1=212.87$
$\beta2=2.04$, $\eta2=5.86$, $\gamma2=36.37$
$\beta3=2.21$, $\eta3=31.98$, $\gamma3=180.70$
$\beta4=2.65$, $\eta4=4.92$, $\gamma4=27.87$

High efficiency



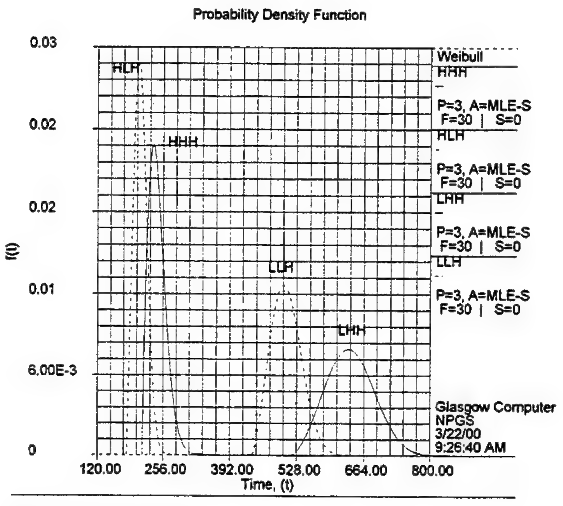
$\beta1=2.80$, $\eta1=139.39$, $\gamma1=513.88$
$\beta2=1.83$, $\eta2=14.07$, $\gamma2=88.39$
$\beta3=2.47$, $\eta3=78.90$, $\gamma3=437.48$
$\beta4=2.87$, $\eta4=14.66$, $\gamma4=74.93$

Low efficiency

In both cases the increment of volatility produces a shift to the right. This shift is magnified when the complexity is high. The effect is also magnified when the efficiency is low.

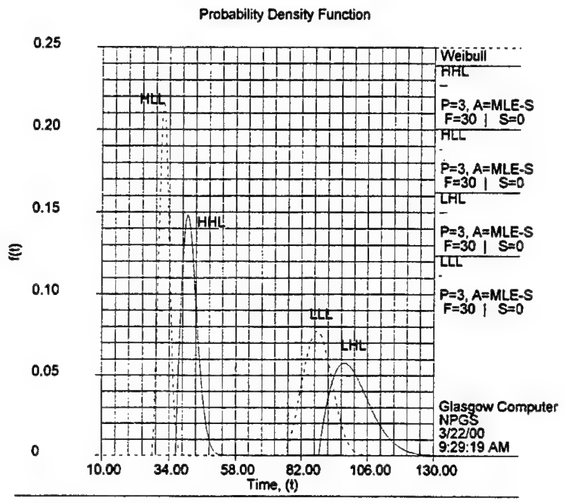
E. Effect of efficiency

The following graphs show the influence of efficiency. Two graphs are presented to discriminate the cases of high and low complexity in order to avoid confounding factors.



β1=1.78, η1=34.98, γ1=212.87
β2=2.21, η2=31.98, γ2=180.70
β3=2.80, η3=139.39, γ3=513.88
β4=2.47, η4=78.90, γ4=437.48

High complexity



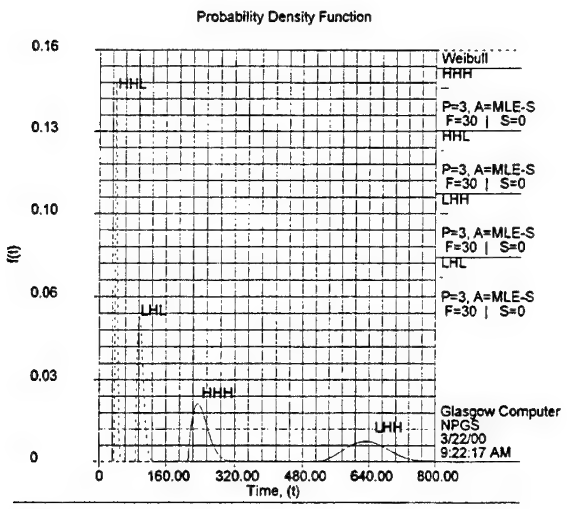
β1=2.04, η1=5.86, γ1=36.37
β2=2.65, η2=4.92, γ2=27.87
β3=1.83, η3=14.07, γ3=88.39
β4=2.87, η4=14.66, γ4=74.93

Low complexity

In both cases the increment on efficiency produces a shift to the left. This shift is magnified when the complexity is high. The effect is also magnified when the volatility is high.

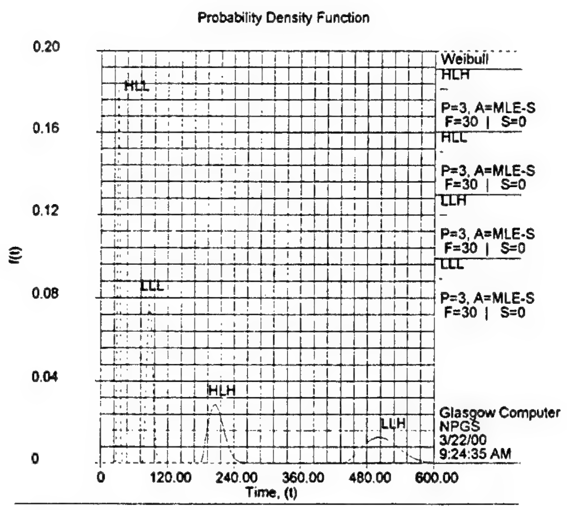
F. Effect of complexity

The following graphs show the influence of complexity. Two graphs are presented to discriminate the cases of high and low requirements volatility in order to avoid confounding factors.



$\beta1=1.78, \eta1=34.98, \gamma1=212.87$
$\beta2=2.04, \eta2=5.86, \gamma2=36.37$
$\beta3=2.80, \eta3=139.39, \gamma3=513.88$
$\beta4=1.83, \eta4=14.07, \gamma4=88.39$

High volatility



$\beta1=2.21, \eta1=31.98, \gamma1=180.70$
$\beta2=2.65, \eta2=4.92, \gamma2=27.87$
$\beta3=2.47, \eta3=78.90, \gamma3=437.48$
$\beta4=2.87, \eta4=14.66, \gamma4=74.93$

Low volatility

In both cases the increment on complexity produces a shift to the right. This shift is magnified when the efficiency is low. The effect is also magnified when the volatility is high.

G. Cumulative density functions and stochastic dominance

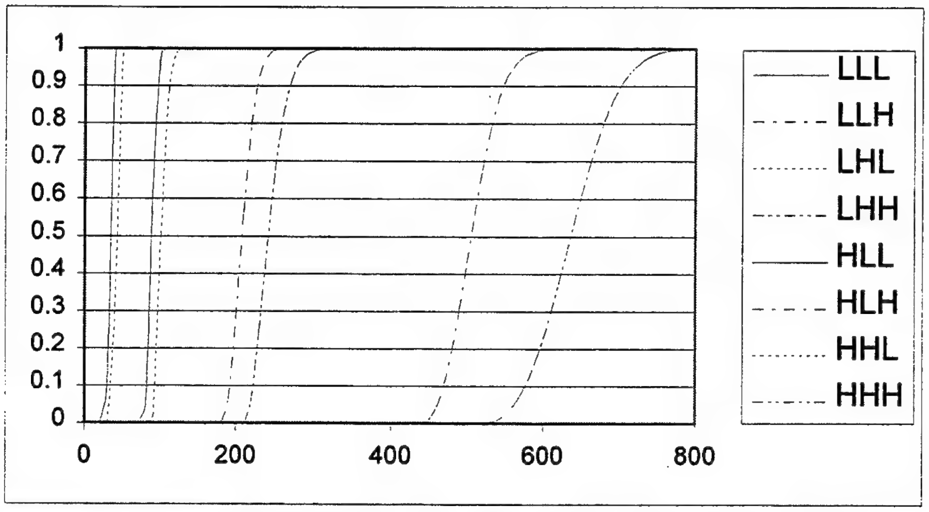


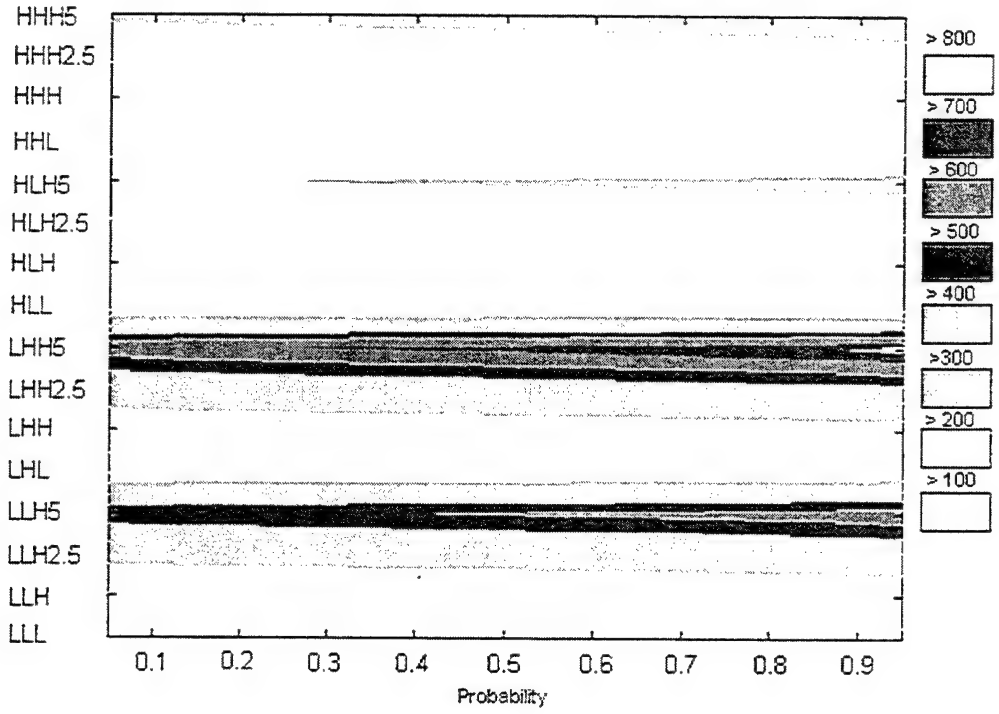
Figure x.4: Comparison of the cdfs for the different scenarios.

As expected, for same level of complexity high efficiency scenarios have stochastic have stochastic dominance.

H. Contour of time

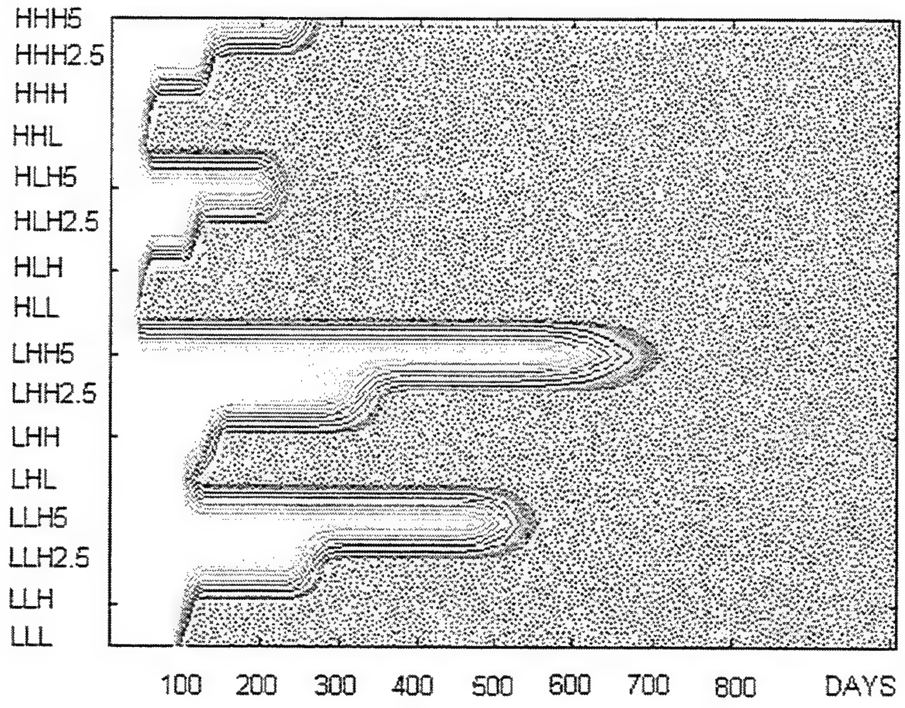
One of the difficulties in visualizing the model is that it has four variables (efficiency, requirements volatility, complexity, and time), hence it is necessary a five dimensional space to represent it (four dimensions for the parameters plus one extra dimension for the scalar value of the probability associated).

The following graph represents the lines of same expected time given a discrete set of scenarios with different efficiency, complexity, and requirements volatility. The graph is only useful to visualize the combined effect of the three parameters of the model. Given a certain scenario and a confidence probability it is possible to determine the expected time in days. For instance, the comparison of HHH5 (high efficiency, high volatility, high complexity) vice LHH5 (low efficiency and the same other parameters) show the effect of efficiency.



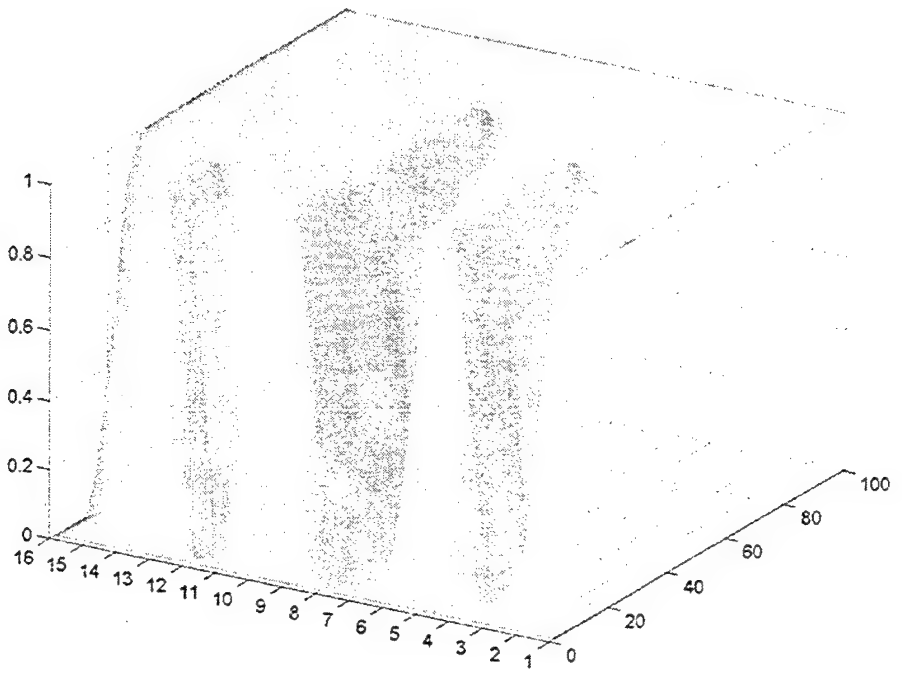
I. Contour of probabilities

The following graph represents the lines of same probability of finishing the project at a given date, given a discrete set of scenarios with different efficiency, complexity, and requirements volatility. The graph is only useful to visualize the combined effect of the three parameters of the model.



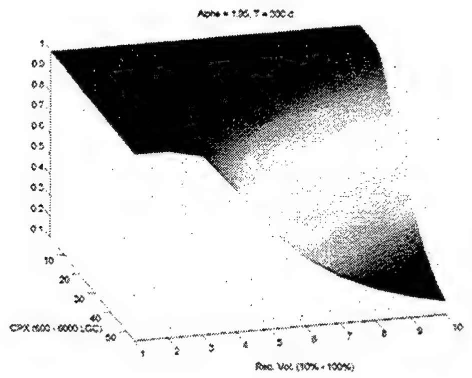
J. Surface of cumulative distribution

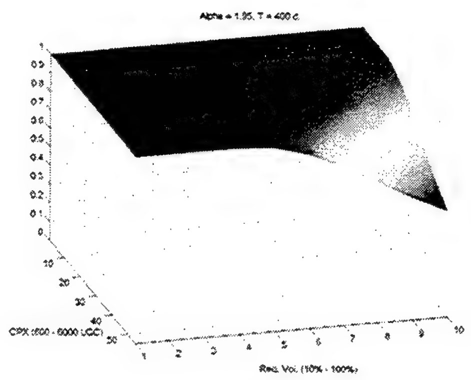
The following graph represents a 3D view of the cumulative distributions for a discrete set of scenarios. The z-axis represents the cdf, the x-axis represents the scenario (1-16), and the y-axis represents the time (0-100).

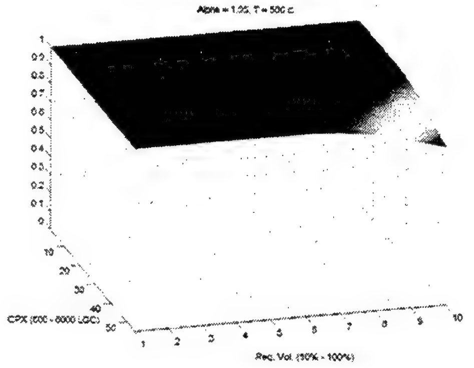


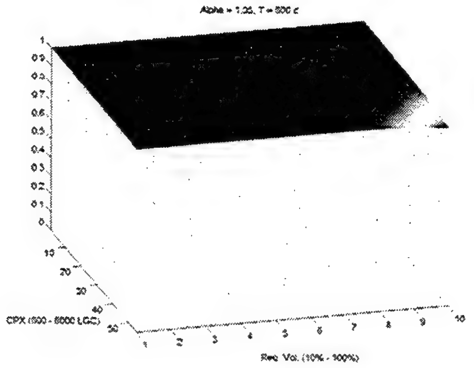
K. Snapshots of the surface of cumulative distribution for high efficiency

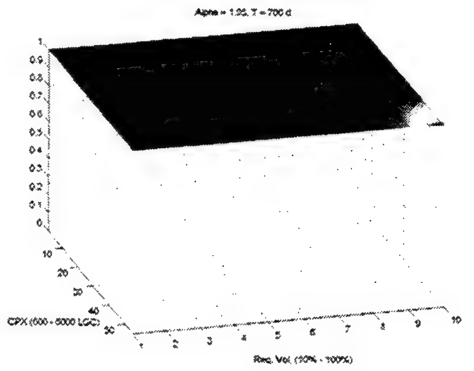
The following series of graphs represents the continuous 3D aspect of the five-dimension model given a high efficiency scenario for five different moments in time. The axes represent complexity, volatility, and cdf. The five snapshots represent time in a discrete way. Efficiency is constant and high for all the graphs.





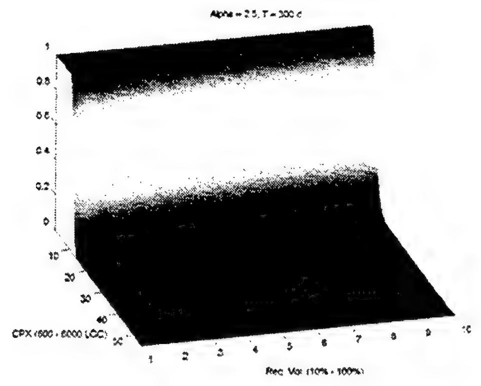


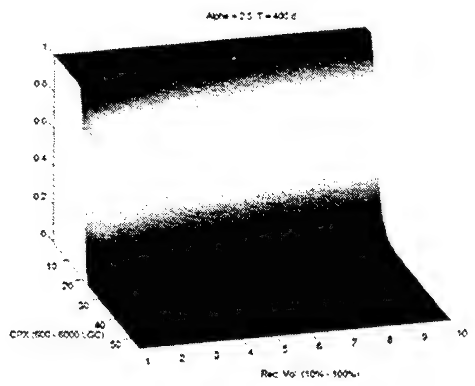


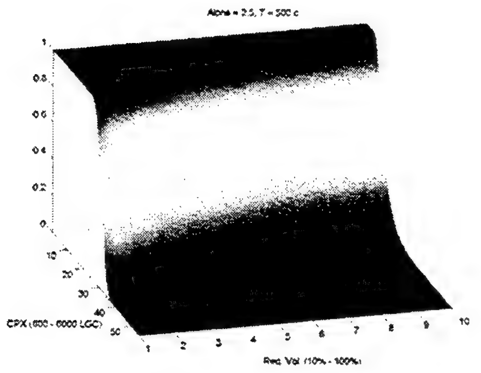


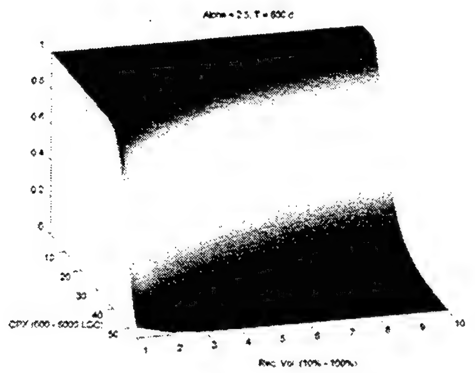
L. Snapshots of the surface of cumulative distribution for high efficiency

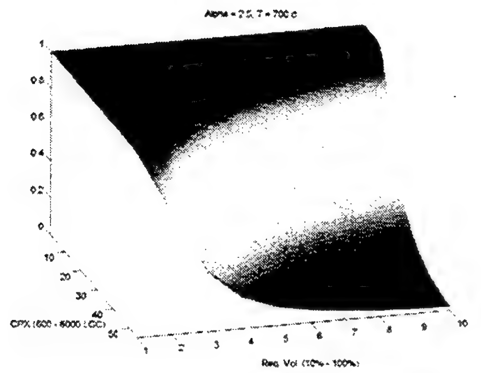
The following series of graphs represents the continuous 3D aspect of the five-dimension model given a high efficiency scenario for five different moments in time. The axes represent complexity, volatility, and cdf. The five snapshots represent time in a discrete way. Efficiency is constant and low for all the graphs.





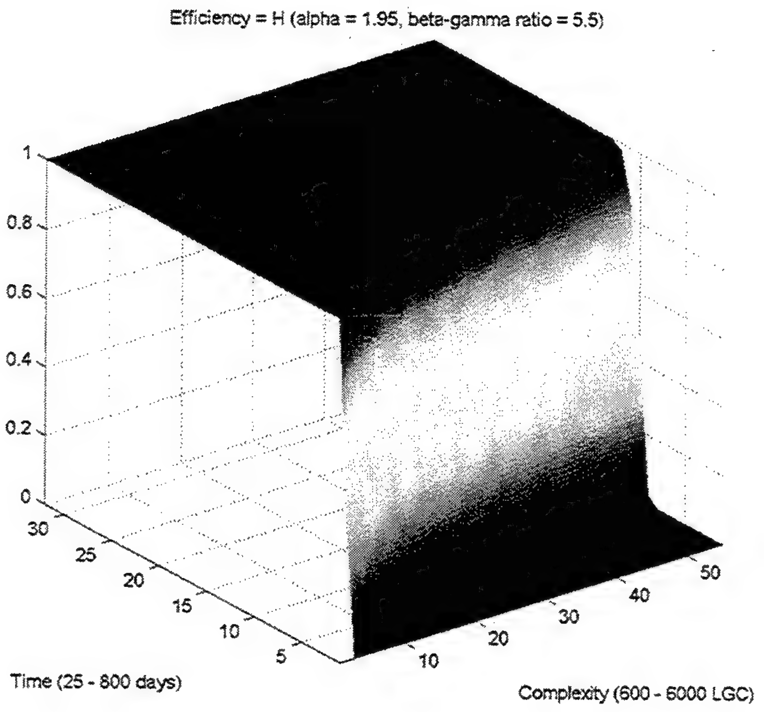






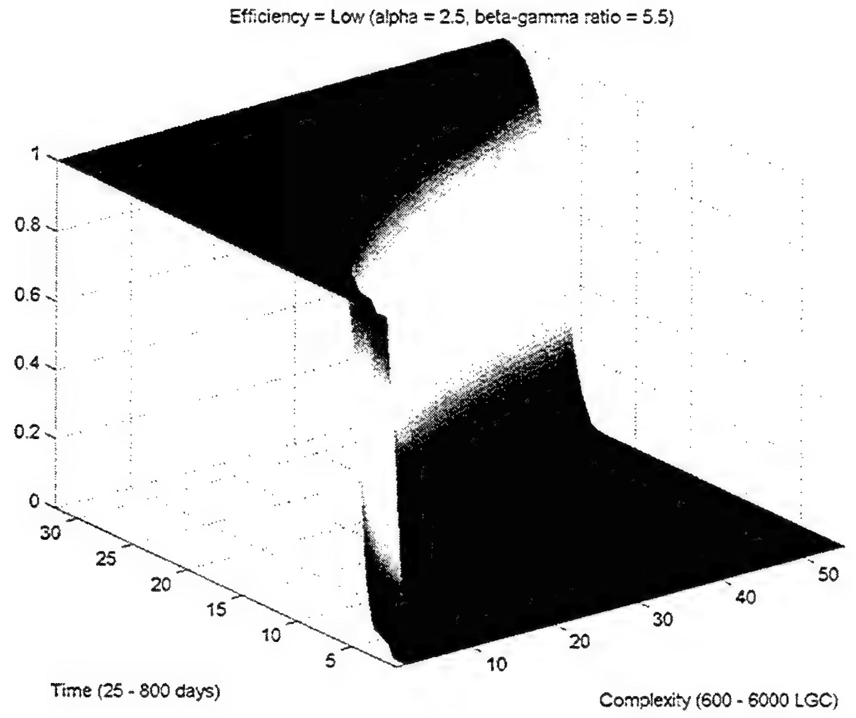
M. Surface of cumulative distribution for high efficiency and gamma-beta ratio = 5.5

The following graph represents the cdf surface for a given level of efficiency and a given level of volatility. The three axes correspond to complexity, time, and cdf. This graph predicts the future of the project under the hypothesis of constant volatility and high efficiency.



N. Surface of cumulative distribution for low efficiency and gamma-beta ratio = 5.5

The following graph represents the cdf surface for a given level of efficiency and a given level of volatility. The three axes correspond to complexity, time, and cdf. This graph predicts the future of the project under the hypothesis of constant volatility and low efficiency.



APPENDIX F

STOCHASTIC DOMINANCE

The major areas of application of dominance have been finance, insurance, and economics. The classical portfolio problem was the catalyst for the initial research. From there the technique was applied to other domains (Whitmore & Findley, 1978). Stochastic dominance is a methodology related to decision theory. It is based on formal concepts and theorems and employs partial information on the decision-maker's preferences and the random variables to produce a partial ordering (Levy, 1998).

Definition of dominance: Let D be a domain constituted by a set of decisions. Let $d \in D$. We say that the decision d_i dominates the domain D (expressed as d_i DOM D), if and only if the return value for the application of d is maximum for all possible values of x and for all possible $dj \in D$.

$(\forall d \in D)(\forall x \in X)(R(x, d_i) \geq R(x, d_j)) \Leftrightarrow d_i \text{ DOM } D$

where D = set of alternatives or decisions, also called Feasible Set (F.S.)
X = set of possible values for the random value x.
R(x, d) = a function that measures the outcome of the decision.

Definition of Efficient Set (E.S.): E.S. is the set of dominating decisions.
$(\forall d \in D)$ $(d \text{ DOM } D) \Rightarrow d \in$ E.S.

Definition of Inefficient Set (I.S.): I.S. is the set of dominated decisions.
F.S. = E.S. $\cup$ I.S.

Definition of First Degree Stochastic Dominance (FSD): FSD is the dominance that can be established by the application of the following Theorem:

Let F(x) and G(x) be cumulative distribution functions (cdf) related to the decisions f and g respectively. We say that f dominates FSD g (f FSD g) if and only if the $F(x) \geq G(x)$ for all values of x.

$(\forall x)\ (F(x) \geq G(x)) \Leftrightarrow (f \text{ FSD } g)$

Observations:

(1) FSD requires that distributions do not intercept, but can be tangent.

(2) When more than two alternatives exist, the mere condition of being dominated by one alternative is sufficient condition to belong to I.S.

(3) All alternatives in E.S. must intercept, and should not be dominated.

Figure G.1 shows an example of inexistence of FSD. cdf1 and cdf2 belong to E.S. cdf3 is clearly dominated so it belongs to I.S. Note that neither cdf1 or cdf2 dominates each other.

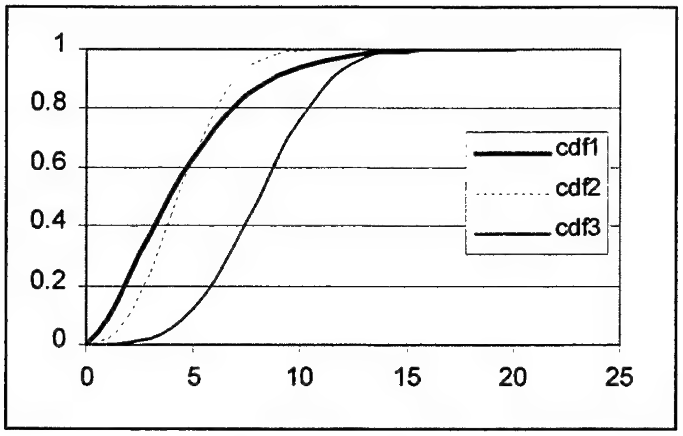


Figure G.1: Concept of domination. cdf1 dominates cdf3

Definition of sufficient conditions for FSD: Let f, g be two alternatives related to F(x) and G(x) respectively.

(1) We say that f dominates first degree stochastic g (f FSD g), if the maximum range of F(x) is less or equal the minimum range of G(x) (Fig. G.2).

$\mathrm{Max}(\mathrm{Range}(F(x))) \leq \mathrm{Min}(\mathrm{Range}(G(x))) \Leftrightarrow (f \text{ FSD } g)$

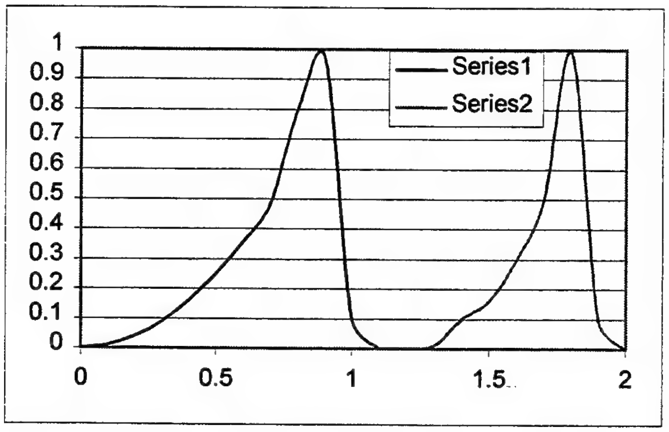


Figure G.2: FSD sufficient condition. Series1 dominates Series2.

(2) We say f dominates first degree stochastic g (f FSD g), if for all values of x F(x) is greater or equal to G(x) (Fig. G.3).

$(\forall x \in X)(F(x) \geq G(x)) \wedge (\exists y \in X)(F(y) > G(y))$

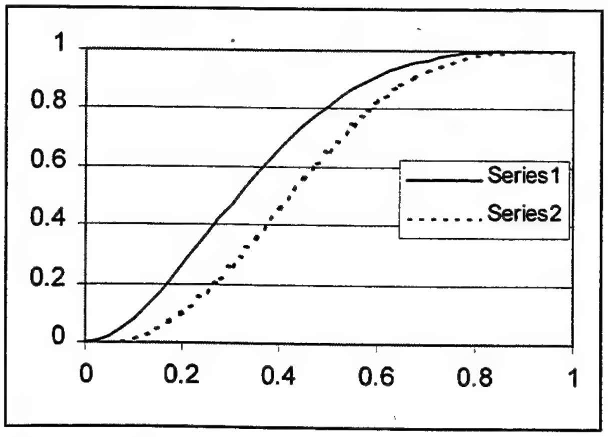


Figure G.3: FSD, second sufficient condition. Series1 dominates Series2.

Definition of Second Degree Stochastic Dominance (SSD): SSD is the dominance that can be established by the application of the following Theorem:

Let f, g be two alternatives with cdf F(x) and G(x) respectively. We say that f dominates g on 2nd degree stochastic dominance (f SSD g), if and only if the area between the two curves is positive.

$(f \text{ SSD } g) \Leftrightarrow \int [F(x) - G(x)]\, dx \geq 0$

Observation

(1) Figure G.3 represents SSD.

(2) Figure G.1 also represents SSD of cdf2 over cdf1 if we assume that the area under cdf2 is greater than the area under cdf1.

Definition of sufficient conditions for SSD: FSD is sufficient for SSD.

$(f \text{ FSD } g) \Rightarrow (f \text{ SSD } g)$

Definition of Third Degree Stochastic Dominance (TSD): The third degree of stochastic dominance is the preference for positive skewness on the pdfs. The skewness (γ) is defined as the ratio of the third moment over the standard deviation to the third.

$\gamma = [\int f(x)\,(x - \mu)^3\, dx\,] / \sigma^3$

Definition of the sufficient conditions for TSD:

(1) FSD is sufficient for TSD.

(2) SSD is sufficient for TSD.

LIST OF REFERENCES

(Abdel-Hamid, 1989)	Abdel-Hamid, T. *Lessons learned from modeling the dynamics of Software.* Communications of the ACM. December, 1989.
(Abdel-Hamid, 1991)	Abdel-Hamid, T. *Software Project Dynamics: An Integrated Approach.* Prentice Hall, 1991.
(Agresti, 1992)	Agresti and Evanco. *Projecting Software Defects from Analyzing Ada Designs.* IEEE Transactions on Software Engineering, vol 18 no. 11, November 1992, pp. 988-997.
(AIAA, 1993)	American Institute of Aeronautic and Astronautics. *Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992, February 1993.
(ANSI, 1991)	ANSI/IEEE *Standard Glossary of Software Engineering Terminology*. STD-729-1991.
(Albrecht, 1979)	Albrecht, A. *Measuring Application Development Productivity.* Proceedings IBM. October 1979.
(Albrecht, 1983)	Albrecht, A. and Gaffney, J. *Software Function Source Lines of Code and Development Effort prediction.* IEEE Transactions on Software Engineering, SE-9, 1983.
(Badr, 1993)	Badr, S. *A Model and Algorithms for a Software Evolution Control System.* PhD Dissertation, Computer Science Department. Naval Postgraduate School. Monterey, CA. 1993.
(Baligh et al., 1994)	Baligh, H., Burton, R., and Obel, B. Validating an Expert System that Designs Organizations. In Computational Organization Theory edited by Carley, K. and Prietula, M. Lawrence Erlbaum Associates, Publishers. 1994

(Baligh et al., 1996)	Baligh, H., Burton, R., and Obel, B. *Organizational Consultant: Creating a Useable Theory for Organizational Design.* Management Science, 42(12). 1996.
(Baybutt, 1989)	Baybutt, P. *Uncertainty in Risk Analysis. Mathematics in Major Accidents Risk Analysis.* Edited by R.A. Cox. Clarendon Press - Oxford, 1989.
(Berzins, 1990)	Berzins, V. and Luqi. *Software Engineering with Abstractions*. Addison-Wesley, 1990.
(Boehm, 1981)	Boehm, B. *Software Engineering Economics.* Prentice Hall, 1981.
(Boehm, 1984)	Boehm, B. *Verifying and Validating Software Requirements and Design Specifications.* IEEE Software, January 1984.
(Boehm, 1988)	Boehm, B. *A Spiral Model of Software Development and Enhancement. Computer.* May, 1988.
(Boehm, 1988a)	Boehm, B. and Belz, F. *Applying Process Programming to the Spiral Model.* Proceedings of the 4th International Software Process Workshop. May 1988.
(Boehm, 1989)	Boehm, B. *Software Risk Management.* IEEE Computer Society Press. 1989.
(Boehm, 1991)	Boehm, B. *Software Risk Management: Principles and Practices.* IEEE Software, January, 1991.
(Boehm, 1997)	Boehm, B. & De Marco T. *Software Risk Management.* IEEE Software. May-June, 1997.
(Boehm, 2000)	Boehm, B., Madachy R., Selby, R. *Cost Models for Future Software Life Cycle Processes: COCOMO 2.0.* http://sunset.usc.edu/COCOMOII/cocomo.html
(Brooks, 1974)	Brooks, F. *The Mythical Man-Month.* Datamation. December. 1974.

(Brown & Eisenhardt, 1998) Brown, S. and Eisenhardt, K. *Competing on the Edge. Strategy as Structured Chaos.* Harvard Business School Press. 1998.

(Burton & Obel, 1998) Burton, R., and Obel, B. *Strategic Organizational Diagnosis and Design. Developing Theory for Application.* Kluwer Academic Publishers. 1998.

(Campbell & Stanley, 1966) Campbell, D. and Stanley, J. Experimental and Quasi-experimental Designs for Research. Rand McNally, 1966

(Carr, 1997) Carr, M. *Risk Management May not be for Everyone.* IEEE Software, May - June 1997.

(Charette, 1997) Charette, R., Adams, K., & White, M. *Managing Risk in Software Maintenance.* IEEE Software, May-June, 1997.

(Chen, 1978) Chen, E. *Program Complexity and Programmer Productivity.* IEEE Trans. Soft. Eng. May 1978.

(Christiansen, 1993) Christiansen, T. R. *Modeling the Efficiency and Effectiveness of Coordination in ·Engineering Design Teams.* Ph.D. Dissertation, Department of Civil Engineering, Stanford University. Published as Det Norske Veritas Research Report No. 93-2063, Oslo, Norway.

(Chulani et al., 1999) Chulani, Sunita Boehm, Steece. *Bayesian Analysis of Empirical Software Engineering Cost Models.* IEEE Transactions on Software Engineering. July-August, 1999.

(Cohen, 1992) Cohen G.P. *The Virtual Design Team: An Object-Oriented Model of Information Sharing in Project Teams* [Ph.D.]. Stanford: Stanford University, 1992.

(Conklin, 1988) Conklin, J. and Begeman, M. *GIBIS: A Hypertext Tool for Exploratory Policy Discussion.* ACM Transactions on Office Information Systems. Vol. 6. October, 1988.

(Conte, 1986) Conte. S, Dunsmore, H. and Shen, V. *Software Engineering Metrics and Models.* Benjamin Cummings. 1986.

(Cook & Campbell, 1976)	Cook, T., Campbell, D. *The Design and Conduct of Quasi-Experiments and True Experiments in Field Settings.* In *Handbook of Industrial and Organizational Psychology.* Rand-McNally. Dunnette (editor). 1976.
(Cullen & Frey, 1999)	Cullen, A. and Frey, H. *Probabilistic Techniques in Exposure Assessment. A Handbook for Dealing with Variability and Uncertainty in Models and Inputs.* Plenum Press. 1999.
(Cusumano, 1999)	Cusumano, M. and Yoffie, D. *Software Development on Internet Time.* Computer. October, 1999.
(Dalkey & Helmer, 1963)	Dalkey, N and Helmer, O. *An Experimental Application of the Delphi Method to the Use of Experts.* Management Science, 1963, 9, 458-467.
(Devore, 1995)	Devore, J. *Probability and Statistics for Engineering and the Sciences.* Duxbury. 1995.
(Dooley, 1994)	Dooley, K. and Flor, R. *Success and Failure in Total Quality Management Initiatives.* Proceeding of the Chaos Network, Denver, 1994.
(Elsayed, 1996)	Elsayed, E. *Reliability Engineering.* Addison Wesley. 1996.
(Fenton & Pfleeger, 1997)	Fenton, N. and Pfleeger, S.L. Software Metrics. A Rigorous & Practical Approach. PWS Publishing Co. 1997.
(Field, 1997)	Field, T. *When BAD Things Happen to GOOD Projects.* CIO, 15 October, 1997.
(Gaffney, 1988)	Gaffney and Davis. *An Approach to Estimating Software Errors and Availability.* SPC-TR-88-007 version 1.0, March 1988. Proceedings of the Workshop on Software Reliability, July 1988.
(Galbraith, 1977)	Galbraith, J. R., *Organization Design. Reading,* MA: Addison-Wesley, 1977.

(Garvey, 1997)	Garvey, P., Phair, D. and Wilson, J. *An Information Architecture for Risk Assessment and Management.* IEEE Software, May - June 1997.
(Gemmer, 1997)	Gemmer. *Risk Management: Moving Beyond Process.* Computer Vol. 30 Issue 5. May, 1997.
(Gilb, 1977)	Gilb, T. *Software Metrics.* Winthrop Publishers, Inc. 1977.
(Gilb, 1988)	Gilb, T. *Principles of Software Engineering Management.* Addison-Wesley 1988.
(Hall, 1997)	Hall, E. Managing Risk. *Methods for Software Systems Development.* Addison Wesley, 1997.
(Harn, 1999a)	Harn, M., Berzins, V. and Luqi. *Software Evolution via Reusable Architecture.* Proceedings of 1999 IEEE Conference and Workshop on Engineering of Computer-Based Systems. Nashville, Tennesee. March, 1999.
(Harn, 1999b)	Harn, M. *Computer-Aided Software Evolution Based on Inferred Dependencies.* Proceedings of Conference on Advanced Information Systems Engineering: 6th Doctoral Consortium. Heidelberg, Germany. June, 1999.
(Harn, 1999c)	Harn, M., Berzins, V. and Luqi. *A Dependency Computing Model for Software Evolution.* Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering. Kaiserslautern, Germany. June, 1999.
(Harn, 1999e)	Harn, M., Berzins, V. and Luqi. *Computer-Aided Software Evolution Based on a Formal Model.* Proceedings of the 13th International Conference on Systems Engineering. Las Vegas, Nevada. August, 1999.
(Harn, 1999f)	Harn, M. *Relational Hypergraph Model.* PhD Disssertation. Naval Postgraduate School. Monterey, California. 1999.
(Huberman & Glance, 1998)	Huberman, B. and Glance, N. Fluctuating Efforts and Sustainable Cooperation. Chapter 5 on Prietula,

	M., Carley, K., Gasser L. *Simulating Organizations. Computational Models for Institutions and Groups.* MIT Press, 1998.
(Humphrey, 1987)	Humphrey, W. et al. *A Method for Assessing the Software Capability of Contractors.* CMU/SEI-87-TR-23. 1987.
(Humphrey, 1989)	Humphrey, W. *Managing the Software Process.* Addison-Wesley, 1989.
(Ibrahim, 1996)	Ibrahim, O. *A Model and Decision Support Mechanism for Software Requirements Engineering.* Ph.D. Dissertation. Naval Postgraduate School. Monterey, California. 1996.
(James, 1996)	James, G. E. *Chaos Theory. The Essentials for Military Applications.* Naval War College. The Newport Papers, 1996.
(Johnson, 1994)	Johnson, N., Kotz, S., and Balakrishnan N. *Continuous Univariate Distributions. Vol. 1.* Wiley & Sons, 1994.
(Jones, 1994)	Jones, Capers. *Assessment and Control of Software Risks.* Yourdon Press Prentice Hall, 1994.
(Jones, 1996)	Jones, Capers. *By Popular Demand: Software Estimating Rules of Thumb.* Computer, March 1996.
(Jin, 1996)	Jin, Y. and Levitt, R. (Department of Civil Engineering, Stanford University*). The Virtual Design Team: A Computational Model of Project Organizations.* Paper to appear in Computational and Mathemetical Organization Theory. 1996.
(Kang et al., 1998)	Kang, M., Waisel, L. and Wallace, W. *Team Soar. A Model for Team Decision Making*. Chapter 2 on Prietula, M., Carley, K., Gasser L. *Simulating Organizations. Computational Models for Institutions and Groups.* MIT Press, 1998.
(Karolak, 1996)	Karolak, D. *Software Engineering Management.* IEEE Computer Society Press, 1996.

(Kauffman, 1995) Kauffman, Stuart. *At Home in the Universe.* Oxford University Press, 1995.

(Kemerer, 1993) Kemerer, C. *Reliability of Function Points Measurements: A Field Experiment.* Communications of ACM, Vol 36 No 2. 1993.

(Kemerer, 1997) Kemerer, C. *Software Project Management. Readings and Cases*. McGraw-Hill. 1997

(Kitchenham, 1993) Kitchenham, B., Kansala, K. *Inter-item Correlations among Function Points.* First International Software metrics Symposium. IEEE Computer Society Press. 1993.

(Kitchenham, 1997) Kitchenham, B., Linkman, S. *Estimates, Uncertainty, and Risk.* IEEE Software. May-June, 1997.

(Kunz et al., 1998) Kunz, J. C., Tore R. Christiansen, Geoff P. Cohen, Yan Jin, Raymond E. Levitt. *The Virtual Design Team: A Computational Simulation Model of Project Organizations*. Communications of the Association for Computing Machinery (CACM) 41 (11), November, 1998, pp. 84-91.

(Levitt et al., 1994) Levitt, R. E., G. P. Cohen, J. C. Kunz, C. I. Nass, T. Christiansen, and Y. Jin (1994), *The Virtual Design Team: Simulating How Organization Structures and Information Processing Tools Affect Team Performance,* in K. Carley and M. Prietula (Eds.) Computational Organizational Theory, Hillsdale, NJ: Lawerence Erlbaum Associates.

(Levitt, 1999) Levitt, R. *The ViteProject Handbook: A User's Guide to Modelling and Analyzing Project Work Processes and Organizations.* Vité ©. 1999.

(Levitt, 2000) Levitt, R. *VDT Computational Emulation Models of Organizations: State of the Art and the Practice.* Center for Integrated Facility Engineering. Stanford University, 2000.

(Lin, 1998) Lin, Z. *The Choice Between Accuracy and Errors. A Contingency Analysis of External Conditions and Organizational Decision Making Performance.*

	Chapter 4 on Prietula, M., Carley, K., Gasser L. *Simulating Organizations. Computational Models for Institutions and Groups.* MIT Press, 1998.
(Levy, 1998)	Levy, H. *Stochastic Dominance. Investment Decision Making under Uncertainty.* Kluwer Academic Publishers. 1998.
(Londeix, 1987)	Londeix, B. *Cost Estimation for Software Development.* Addison-Wesley, 1987.
(Lorenz, 1995)	Lorenz, Kidd. *OO Software Metrics.* Prentice Hall, 1995.
(Luqi, 1988a)	Luqi and Ketabchi, M. *A Computer-Aided Prototyping System.* IEEE Software. March, 1988.
(Luqi, 1988b)	Luqi and Berzins, V. *Rapidly Prototyping Real-Time Systems.* IEEE Software. September, 1988.
(Luqi, 1989)	Luqi. Software Evolution *Through Rapid Prototyping.* IEEE Computer. May, 1989.
(Luqi, 1990)	Luqi. *A Graph Model for Software Evolution.* IEEE Transactions on Software Engineering. Vol. 16 No. 8. August, 1990.
(Luqi)	Luqi. *Formal Models and Prototyping.* Research supported by National Science Foundation (CCR-9058453) and by the Army research Office (30989-MA).
(Luqi, 1991)	Luqi and Royce, W. *Status Report: Computer-Aided Prototyping.* IEEE Software. November, 1991.
(Luqi, 1997)	Luqi and Goguen, J. *Formal Methods: Promises and Problems.* IEEE Software. January, 1997.
(Lyu, 1995)	Lyu, M. *Software Reliability Engineering.* IEEE Computer Society Press. 1995.
(McFarlan, 1974)	McFarlan, F. *Portfolio Approach to Information Systems.* Harvard Business Review. January-February, 1974.

(Marshall, 1995)	Marshall, K. Oliver, R. *Decision Making and Forecasting.* McGraww-Hill, 1995.
(Mostov, 1989)	Mostov, Luqi and Hefner. *A Graph Model of Software Maintenance.* Technical Report NPS52-90-014. Department of Computer Science. Naval Postgraduate School. Monterey, CA. August 1989.
(Mostov, 1989)	Mostov. *A Model of Software Maintenance for Large Scale Military Systems.* Master's Thesis. Naval Postgraduate School. Monterey, CA. June, 1990.
(Munson, 1995)	Munson, J. and Khoshgofar, T. Chapter 12 (Lyu, 1995).
(Musa, 1998)	Musa, J. *Software Reliability Engineering: More Reliable Software, Faster Development and Testing.* McGraw-Hill, 1998.
(Myers, 1976)	Myers, G. *Software Reliability.* John Wiley & Sons. 1976.
(Nissen, 1998)	Nissen, M. *Redesigning Reengineering through Measurement-Driven Inference.* MIS Quarterly. December, 1998.
(Nogueira et al., 2000a)	Nogueira, J.C., Luqi, and Berzins, V. *A Formal Risk Assessment Model for Software Evolution.* SEKE 2000. Chicago, July 2000.
(Nogueira et al., 2000b)	Nogueira, J.C., Luqi, and Bhattacharya, S. *A Risk Assessment Model for Software Prototyping Projects.* IEEE Workshop on Rapid System Prototyping RSP 2000. Paris, June 2000.
(Nogueira et al., 2000c)	Nogueira, J.C., Luqi, , Berzins, V., and Nada, N. *A Formal Risk Assessment Model for Software Evolution.* ICSE 2000. Limerick, June 2000.
(Nogueira et al., 2000d)	Nogueira, J.C., Luqi, and Berzins, V. *Risk Assessment in Software Requirement Engineering.* IDTP 2000. Dallas, June 2000.

(Nogueira, 2000e)	Nogueira, J.C., Jones, C. R., and Luqi. *Surfing on the Edge of Chaos: Applications to Software Engineering.* CCRP 2000. Monterey, June 2000.
(Norden, 1963)	Norden, Peter. *Resource Usage and Network Planning Techniques.* In Operations Research in Research and Development. Edited by Dean, B. John Wiley & Sons 1963 pp. 149-169.
(O'Leary, 1988)	O'Leary, D. *Methods of Validating Experts Systems.* Interfaces #18. 1988.
(Porter, 1980)	Porter, Michael. *Competitive Strategy.* Free Press, 1980.
(Pfleeger, 1999)	Pfleeger, S.L. *Albert Einstein and Empirical Software Engineering.* IEEE Computer. October 1999.
(Pressman, 1992)	Pressman. *Software Engineering,* 1992
(Prietula et al, 1998)	Prietula, M., Carley, K., Gasser L.Simulating Organizations. Computational Models for Institutions and Groups. MIT Press, 1998.
(Putnam, 1980)	Putnam, L. *Software Cost Estimating and Life-cycle Control: Getting the Software Numbers.* IEEE Computer Society Press. 1980.
(Putnam, 1992)	Putnam, L. and Myers, W. *Measures for Excellence. Reliable Software On Time Within Budget.* Yourdon Press, 1992.
(Putnam, 1996)	Putnam, L. and Myers, W. *Executive Briefing. Controlling Software Development.* IEEE Computer Society Press. 1996.
(Putnam, 1997)	Putnam, L. and Myers, W. *Industrial Strength Software.* Effective Management Using Measurement. IEEE Computer Society Press, 1997.
(Ramesh, 1992)	Ramesh, B. and Dhar, V. *Supporting Systems Development Using Knowledge Captured During Requirements Engineering.* IEEE Transactions on Software Engineering. June, 1992.

(Ramesh, 1995) Ramesh and Luqi. *An Intelligent Assistant for Requirements Validation.* Journal of Systems Integration, 5, 157-177. 1995.

(Reel, 1999) Reel, J. *Critical Success Factors in Software Projects.* IEEE Software. May - June, 1999.

(Render, 1997) Render, B. and Stair, R. *Quantitative Analysis for Management*. Prentice Hall, 1997.

(Rifkin, 2000) Rifkin, S. *When the Project Absolutely Must Get Done: Marrying the Organization Chart with the Precedence Diagram.* International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 2000.

(Roos, 1996) Roos, Johan. *The Poised Organization: Navigating Effectively on Knowledge Landscapes.*, 1996. http://www.imd.ch/fac/roos/paper_po.html

(Santosus, 1998) Santosus, Megan. *Simple, Yet Complex.* Business Management CIO Enterprise Magazine. April 15, 1998.

(SEI, 1996) Software Engineering Institute. *Software Risk Management.* Technical Report CMU/SEI-96-TR-012. June, 1996.

(Schneidewind, 1975) Schneidewind, N. *Analysis of Error Processes in Computer Software.* Proceedings of the International Conference on Reliable Software. IEEE Computer Society, 21-23 April 1975. Pp 337-346.

(Senegupta and Jones, 1999) Sengupta, K. and Jones Carl R. *Creating Structures for Network-Centric Warefare: Perspectives from Organizational Theory.* Command & Control Research & Technology Symposium. CCRP 1999. Naval War College, 1999.

(Sommerville, 1992) Sommerville, I. *Software Engineering.* 1992

(Thomsen et al., 1999) Thomsen, Jan, Raymond E. Levitt, John C. Kunz, Clifford I. Nass, Douglas B. Fridsma. *A Trajectory for Validating Computational Emulation Models of*

	Organizations. Journal of Computational & Mathematical Organization Theory, 5, (4), December 1999, pp. 385-401
(Turban & Aronson, 1998)	Turban, E. and Aronson, J. *Decision Support Systems and Intelligent Systems.* Prentice Hall. 1998.
(USAF, 1988)	USAF. *Software Risk Abatement.* ASFC/AFLC pamphlet 800-45, US Air Force Systems Command. Andrews AFB. 1988.
(vanGenuchten, 1991)	van Genuchten, M. *Why is Software Late? An Empirical Study of the Reasons for Delay in Software Development.* IEEE Transactions on Software Engineering. June, 1991.
(von Bertalanfy, 1976)	von Bertalanfy, L. *General System Theory: Foundations, Development, Applications.* Braziller, 1976.
(Walston, 1977)	Walston, C. and Felix, C. *A Method of Programming Measurement and Estimation.* IBM Systems Journal. Vol. 16 No. 1, 1977.
(Weibull, 1939)	Weibull, W. *A Statistical Theory of the Strength of Material. Report No. 151*, Ingeniors Vetenskaps Akademiens Handligar. Stockholm, 1939.
(Whitmore & Findlay, 1978)	Whitmore, G. and Findlay, M. *Stochastic Dominance.* Lexington Books. 1978.
(Wideman, 1992)	Wideman, R. *Risk Management. A Guide to Managing Project Risk Opportunities.* Project Management Institute. 1992.
(Woodward, 1965)	Woodward, J. *Industrial Organization Theory and Practice.* Oxford University Press. 1965.
(Woodward, 1999)	Woodward, S. *Evolutionary Project Management.* IEEE Computer, October 1999.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John j. Kingman Rd., STE 0094
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5000

3. Dr. Dan Boger... 1
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5000

4. Dr. Carl R. Jones 1
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5000

5. LtCol Terrance Brady.. 1
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5000

6. Comando General de la Armada 10
   Edificio Comando General 4 Piso
   Rambla 25 de Agosto de 1825 S/N
   Montevideo, CP 11000
   Uruguay

7. CAPT Juan C. Nogueira... 5
   Edificio Comando General 4 Piso
   Rambla 25 de Agosto de 1825 S/N
   Montevideo, CP 11000
   Uruguay